Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

TREBALL FINAL DE GRAU

TÍTOL DEL TFG: Estudi d'emuladors de xarxa

TITULACIÓ: Grau en Enginyeria de Sistemes de Telecomunicació

AUTOR: Issam Aouad Tabet

DIRECTOR: Roc Meseguer Pallarès

DATA: 22 de novembre 2019

Títol: Estudi d'emuladors de xarxa

Autor: Issam Aouad Tabet

Director: Roc Meseguer Pallarès

Data: 22 de novembre 2019

Resum

L'emulació de xarxes permet replicar cadascun dels components d'una xarxa, així com el seu comportament, sota unes condicions determinades, i d'acord als recursos i les tecnologies disponibles. En l'actualitat trobem diversos emuladors de xarxa que, responent a diferents necessitats, es presenten com una alternativa real a les xarxes físiques.

En aquest treball es pot trobar un estudi dels principals emuladors de xarxa disponibles en la actualitat, incloent el seu estat actual, la seva arquitectura i les seves funcionalitats. L'objectiu és entendre les tecnologies que fan possible l'emulació de xarxes, diferenciar els diferents casos d'us i explorar el seu potencial.

Partirem de l'emulador CORE per il·lustrar i documentar diferents casos d'ús, fent èmfasis en els nous serveis i en proposar millores per tal d'explotar el potencial de l'emulador en facetes menys conegudes.

CORE és un emulador complet, però amb una clara vocació pel desplegament de xarxes a petita escala utilitzant la GUI. A través de l'estudi i la integració de diferents eines, l'emulador pot ser una alternativa pel desplegament i la configuració de grans xarxes virtuals, però presenta certes limitacions en escalabilitat, experiència d'usuari i incorporació de serveis basats en OpenFlow, protocol clau en les xarxes actuals.

Title: Study of network emulators

Author: Issam Aouad Tabet

Director: Roc Meseguer Pallarès

Date: November 22th, 2019

Overview

Network emulation allows replicating each one of the components of a network, as well as its behavior, under certain conditions, and according to the available resources and technologies. Nowadays, several network emulators are available which, responding to different needs, are presented as a real alternative to physical networks.

This project contains a study of the main network emulators available today, including its current state, its architecture and its functionalities. The goal is to understand the technologies that make possible the emulation of networks, to differentiate between the different use cases and to explore its potential.

We will use CORE emulator to illustrate and document different use cases, emphasizing in new services and proposing improvements in order to exploit the potential of the emulator in lesser-known facets.

CORE is a complete emulator, but with a clear vocation for deploying small-scale networks using the GUI. Through the study and integration of different tools, the emulator can be an alternative to the deployment and configuration of large virtual networks, but it has certain limitations in scalability, user experience and incorporation of services based on OpenFlow, key protocol in current networks.

ÍNDEX

CAPÍTOL 1. Emulació i simulació de xarxes

1.1. Introducció: motivació, estructura i objectius del treball

Aquest projecte va néixer amb l'objectiu d'ajudar al lector a entendre el propòsit, els blocs fonamentals i les aplicacions pràctiques de l'emulació de xarxes. Les tecnologies de virtualització han impactat tots els aspectes de la informàtica i les xarxes telemàtiques, obrint un món de possibilitats que segueix creixent amb el pas dels anys. L'emulació de xarxes no deixa de ser un altre exemple d'aquesta revolució.

En el primer capítol contextualitzarem el fenomen de l'emulació de xarxes, i explicarem el seu propòsit, fent èmfasis en la relació amb la virtualització d'equips. Esclarirem els termes emulació i simulació de xarxes, habitualment utilitzats indistintament, tot i que en aquest treball ens centrarem en l'emulació.

En el segon capítol explicarem els blocs fonamentals de gran part dels emuladors basats en software lliure, a través de les diferents eines de virtualització. L'objectiu és donar a conèixer al lector les diferents eines i conceptes que fan possible l'emulació de xarxes, i que més endavant veurem en acció.

En el tercer capítol veurem una comparativa d'alguns dels emuladors més coneguts en l'actualitat, per tal de donar una idea al lector de les diferents aplicacions d'aquests, ajudant en una possible tria segons les necessitats particulars del lector. En concret, aquesta és una de les necessitats que va donar peu a aquest projecte amb en Roc, el tutor d'aquest treball.

El quart capítol conté la part pràctica del treball, on trobarem diferents experiments que ens ajudaran a posar en pràctica, de la mà de CORE, els diferents casos d'ús d'un emulador. Farem èmfasis en explorar aspectes menys coneguts de l'emulador CORE, proposar millores, documentar els diferents procediments que no apareixen a la documentació oficial i, en definitiva, presentar l'emulació de xarxes com una alternativa i un complement a les xarxes físiques.

1.2. Context: creixement i transformació de xarxes

L'expansió i la reorganització d'una empresa, la introducció d'una nova aplicació a la xarxa, les migracions de serveis al cloud; tots aquests exemples impliquen noves necessitats de connectivitat que han de ser adreçats. Tots aquests reptes comporten l'aparició de nous protocols, equips i solucions, transformant i expandint l'ecosistema que és una xarxa.

Per cada repte trobem diferents equips i solucions, la qual cosa implica habitualment diferents fabricants, amb protocols propietaris que han de coexistir a la xarxa. El nivell de complexitat creix a mesura que evoluciona la xarxa. Per tant, abans d'introduir qualsevol canvi a la xarxa, és necessari abordar prèviament la integració i el comportament amb la resta de l'ecosistema. És habitual, per exemple, trobar errors en el disseny, incompatibilitats o falta de funcionalitats durant el cicle de desplegament d'una aplicació o servei. En la mesura que la qualitat i afinitat dels escenaris de proves s'aproximin al màxim a la realitat, podrem trobar millors prestacions en el producte final.

En aquest sentit, una de les tècniques més utilitzades és la virtualització, que permet l'abstracció dels recursos d'una màquina a través d'una capa denominada hipervisor, dividint els recursos en múltiples entorns. La virtualització és una realitat des de fa anys, a través de la qual s 'ha aconseguit un estalvi en forma de recursos de diferent índole, i s'ha guanyat en temps de resposta i flexibilitat per satisfer necessitats canviants. La virtualització de servidors va ser pensada inicialment per millorar l'ús dels recursos. Amb el temps, ha evolucionat a una solució per guanyar un entorn més àgil en el desplegament de màquines virtuals i aplicacions, ajudant a les organitzacions a complir els seus requisits empresarials i reduir les despeses d'operació. Això aplica, per exemple, a fases experimentals en el disseny d'una aplicació de xarxa, on s'executaran múltiples proves en diversos escenaris, tots sota els mateixos recursos físics gràcies a la virtualització.

A partir del tipus d'hipervisor podem fer una classificació:

- Hipervisor natiu: l'hipervisor es troba directament sobre el hardware, fent de gestor tant d'aquest com del sistema operatiu convidat. Microsoft Hyper-V o VMWare ESX/ESXi són exemples d'hipervisors natius.

- Hipervisor allotjat: l'hipervisor s'executa sobre el sistema operatiu amfitrió o del host, amb el sistema operatiu convidat com un procés. VirtualBox o QEMU són alguns dels exemples més coneguts.

Un altre classificació rellevant seria per tipus de virtualització:

- Virtualització completa: el sistema operatiu convidat està completament desacoblat del maquinari subjacent per la capa de virtualització, per la qual cosa no és conscient de que està sent virtualitzat, i no necessita cap modificació. La virtualització completa ofereix aïllament complet i la millor seguretat per les màquines virtuals, facilitant la migració i portabilitat. És necessària una simulació del hardware del host.

- Paravirtualització: el sistema operatiu convidat és modificat i es comunica amb l'hipervisor per guanyar en rendiment. El manteniment i la portabilitat de les màquines virtuals és un punt negatiu.

- Virtualització assistida per hardware: semblant a la virtualització completa, permet executar el sistema operatiu convidat sense modificar. Utilitza les capacitats de hardware del host.

La necessitat de trobar formes per provar i experimentar amb xarxes, utilitzant les tecnologies de virtualització, va comportar l'aparició de tres fenòmens: les xarxes de proves, la simulació i l'emulació.

Les xarxes de proves són entorns reals on s'utilitzen equips físics interconnectats per estudiar el comportament de les topologies i propietats de la xarxa. Aquests entorns ofereixen un grau de realitat molt elevat, ajustant-se als entorns de desplegament en producció. La contrapartida és el cost elevat i el manteniment que requereix aquest tipus d'infraestructura en un laboratori.

Una altre tècnica àmpliament utilitzada és l'emulació, on replicarem les característiques i comportament d'un equip o escenari en una plataforma no dissenyada inicialment amb tal propòsit. Es tracta d'adequar aquesta plataforma per tal de ser hàbil en la acollida d'un sistema no admès o no compatible inicialment. Habitualment, això resulta en certes limitacions de rendiment.

Les eines d'emulació i simulació de xarxa ens permeten replicar o representar el comportament d'una xarxa, estalviant temps i recursos durant el desplegament en producció. Les eines d'emulació de xarxa utilitzen les capacitats de virtualització del nucli del sistema operatiu o d'un hipervisor per executar els nodes.

1.3. Emulació o simulació?

Els termes emulació o simulació de xarxes són utilitzats indistintament de manera habitual, tot i tenir significats diferents. Ambdós són extremadament útils per provar diferents components i comportaments d'una xarxa.

La simulació de xarxes ens permet representar virtualment el comportament d'un servei o d'una topologia de xarxa, creant un model teòric hàbil per entendre el seu funcionament. Els simuladors utilitzen tècniques matemàtiques i, normalment, estan basats en la introducció d'esdeveniments discrets, causant la interacció amb el sistema simulat.

L'emulació de xarxes replica de manera virtual una xarxa i el seu comportament, consumint els recursos necessaris no només de les eines de virtualització utilitzades, sinó que també de cadascun dels components que formen els escenaris replicats.

Normalment els simuladors són utilitzats en entorns docents i en etapes inicials del desenvolupament d'un protocol o d'una aplicació. Per altre banda, els emuladors ens ajuden a posar a prova el funcionament en etapes finals del

desenvolupament, tot i que el rendiment estarà sotmès a les limitacions del host d'emulació.

1.4. Formació professional i docència

Tal i com hem comentat prèviament, l'ecosistema que forma una xarxa és divers. Per gestionar tots el equips i serveis implicats, és necessària una qualificació que els fabricants proporcionen a través de les certificacions. Algunes de les tècniques més utilitzades pels enginyers en la preparació de les certificacions són les eines de simulació i emulació. GNS3, per exemple, va ser elaborat expressament per estudiar la certificació CCNP de Cisco. Avui en dia trobem diversos emuladors capaços de virtualitzar imatges d'equips de Cisco Systems o Juniper Networks, entre altres.

CAPÍTOL 2. Arquitectura i funcionament d'un emulador

Cada emulador està basat en una arquitectura, tecnologia de virtualització i software de suport diferents. En línies generals, un emulador necessita d'una tecnologia de virtualització per aixecar nodes, alguna forma de connexió entre nodes i eines per a la manipulació de paquets.

En l'actualitat, els emuladors de xarxes ofereixen diferents possibilitats pel que fa els nodes; emular hardware d'un fabricant per carregar una imatge del SO real, aixecar màquines virtuals a través de virtualització completa per tal de fer córrer certes aplicacions o aprofitar la lleugeresa dels contenidors vers les màquines virtuals, són algunes de les opcions més comuns. Per tal de realitzar aquestes operacions, els emuladors combinen diferents tecnologies d'emulació i virtualització.

Alhora de connectar els nodes i manipular els paquets, els emuladors utilitzen diferents solucions open-source compatibles amb el SO present en el sistema. En alguns casos, aquestes eines s'integren amb l'emulador plenament, mentre que en altres casos l'usuari és responsable de fer la integració.

2.1. KVM: màquines virtuals amb virtualització completa

KVM (Kernel-based Virtual Machine) és una eina de virtualització per nuclis Linux. Aprofita les capacitats de la virtualització per hardware (Intel VT or AMD-V), fent del nucli Linux un hipervisor. Les màquines virtuals poden executar sistemes operatius sense modificar, cada una amb hardware virtualitzat propi (targeta de xarxa, disc, memòria, etc.). KVM necessita de màquines amb arquitectura x86 executant nucli Linux, en processadors Intel o AMD amb les extensions de virtualització per hardware (VT i SVM respectivament).

2.2. QEMU: virtualització assistida per hardware

QEMU és una eina d'emulació i virtualització de hardware open-source. Com a emulador, QEMU permet executar sistemes operatius i programes en hardware no dissenyat amb tal propòsit, emulant tots els components de hardware necessaris. Com a eina de virtualització, QEMU pot integrar amb hipervisors com KVM per utilitzar virtualització per hardware, aprofitant les extensions del processador. QEMU aprofita les capacitats de KVM quan executa un sistema convidat amb la mateixa arquitectura que el sistema amfitrió, per aconseguir rendiments natius. QEMU és una de les plataformes més utilitzades per emular les imatges de disc (vmdk o qcow2) d'equips físics de fabricants.

2.3. Docker: contenidors multi-SO

Docker és una plataforma per automatitzar i gestionar el desenvolupament i el desplegament d'aplicacions en contenidors. Cada contenidor és desplegat executant una imatge, que és un paquet executable format per tot allò necessari per fer córrer l'aplicació: codi, llibreries, variables d'entorn, temps d'execució i fitxers de configuració. Els contenidors representen una forma flexible, lleugera, escalable i portable d'executar aplicacions.

2.4. Jails: contenidors a FreeBSD

L'eina jail de FreeBSD permet fer particions d'un sistema basat en FreeBSD, en forma de diversos sistemes independents més petits anomenats jails. Aquest mecanisme permet la creació d'un entorn segur, separat de la resta del sistema. Els processos creats dins d'un jail són limitats al seu entorn. Cada jail és un entorn virtual executat en el host, amb el seus propis fitxers de sistema, processos, usuaris i xarxes, entre altres.

Les vimage representen el conjunt d'un jail amb la seva pila de xarxa, formant així un contenidor, i permetent a cada jail gestionar la seva pròpia pila de xarxa virtual. Això implica que cada jail pot tenir les seves pròpies adreces, interfícies o rutes. Totes les vimage comparteixen els recursos de hardware del host, tals com processador o memòria.

2.4.1. Connectant jails a FreeBSD

Netgraph és un sistema modular per implementar diferents objectes encarregats de realitzar les funcions de xarxa. Aquests objectes, anomenats nodes, es connecten a través de hooks, i es comuniquen per processar dades i implementar protocols definint una sèrie de mètodes. A través dels nodes de Netgraph, es poden representar elements tals com interfícies, switchs o una lan sense fils, així com emular les característiques típiques d'un enllaç.

2.5. Linux containers

LXC (també conegut con a Linux Containers) és una tecnologia de virtualització a nivell del sistema operatiu capaç d'executar múltiples entorns Linux, completament aïllats, en un sol sistema. Això s'aconsegueix mitjançant la combinació de diferents funcions del nucli de Linux: espais de noms (namespaces) i grups de control (cgroups).

Els grups de control, són una característica del nucli Linux que permet organitzar processos en grups, als quals es pot limitar i controlar els recursos utilitzats. Un cgroup, per tant, és un grup de processos vinculats a un conjunt

de límits o paràmetres, tals com memòria de sistema o temps de CPU, definits a través d'un sistema de fitxers cgroup.

Els espais de noms permeten abstraure els recursos d'un sistema per tal de crear instàncies lògiques aïllades d'aquests, cada una accessible per el conjunt de processos d'un espai de noms. Si un procés s'està executant en un espai de noms, només pot veure i comunicar-se amb altres processos del mateix espai de noms. Hi ha diversos tipus d'espais de noms: cgroup, IPC, network, mount, UTC, user i PID.

Un espai de noms de xarxa és lògicament una altre còpia de la pila de xarxa, proporcionant una sèrie de variables de xarxa aïllades: taula de rutes, regles de firewall, interfícies o adreces IP, entre altres.

2.5.1. Connectant LXC a Linux

Els veth (virtual Ethernet devices) són túnels que permeten la connexió de diferents espais de noms de xarxa, així com connectar amb el host d'emulació. En cada extrem d'un veth trobem un punt final, habitualment en forma d'interfície, de manera que els paquets viatgen entre els dos extrems quan estan disponibles.

2.6. Dynamips: emulant equips Cisco

Dynamips és un emulador de hardware Cisco que permet córrer imatges sense modificar. Dynamips va ser desenvolupat per Christophe Fillot, i en la actualitat és mantingut per la comunitat de GNS3 sota llicència GNU GPLv2. Es necessita accés a hardware Cisco o un contracte de suport per tal de disposar de les imatges, tot i que Cisco no dóna suport tècnic per tals efectes.

Les famílies d'equips Cisco compatibles són les següents: 7200, 3600, 2691, 3725, 3745, 2600 i 1700.

2.7. Altres serveis: commutació i encaminament de paquets

2.7.1. Linux Bridge i Open vSwitch

Linux Bridge implementa un subconjunt de la norma IEEE 802.1d, permetent connectar segments Ethernet. Està basat en quatre components:

- Ports: punt final per commutar trames entre dispositius.

- Pla de control: implementa les funcionalitats de Spaninng Tree Protocol per evitar bucles.

- Pla de dades: processa les trames rebudes dels ports i les commuta en base a l'aprenentatge de les adreces MAC.

- Base de dades MAC: inclou informació sobre els equips de la xarxa i la seva localització.

Open vSwitch és una plataforma de switching, sota llicencia Apache 2.0, creada per entorns virtuals. Va ser dissenyat com una solució complementaria al bridge Ethernet de Linux, per tal de donar resposta a les noves dinàmiques de xarxa, tals com els desplegament en múltiples servidors o la compatibilitat amb OpenFlow. Permet integrar amb múltiples plataformes de virtualització, suporta diversos protocols de xarxa i proporciona una forma programable d'automatitzar les funcions d'un switch. Algunes de les característiques d'OVS són:

- Suport per múltiples plataformes de virtualització basades en Linux (KVM, XEN, Virtual Box), així com integració amb plataformes d'orquestració (Openstack).
- 802.1Q VLAN
- LACP
- QoS: permet controlar el tràfic en les interfícies d'entrada i sortida (tassa màxima).
- Túnels: GRE o VXLAN.
- Suport per OpenFlow.

2.7.2. Quagga i XORP

Quagga és una eina open-source que proporciona serveis de routing basats en TCP/IP, incloent implementacions d'OSPF v2 i v3, IS-IS, RIP v1 i v2, RIPng i BGP-4, per plataformes Unix. Quagga permet a un sistema actuar com un router dedicat, actualitzant la taula de routing del nucli amb la informació obtinguda dels diferents dimonis que gestionen els protocols de xarxa.

XORP és una altre plataforma open-source per proporcionar serveis de routing d'una manera modular. Dóna suport a OSPF v2 i v3, BGP-4, RIPng i v2, ISIS, diferents protocols multicast (PIM o IGMP), així com a protocols de gestió (SNMP). Segueix una arquitectura modular, on els diferents mòduls independents corren en diferents processos UNIX i es comuniquen a través d'APIs.

CAPÍTOL 3. Emuladors de xarxa

A partir d'algunes de les tecnologies i eines anteriors, es defineixen l'arquitectura i les funcionalitats de gran part dels emuladors basats en software lliure. Segons la familiaritat de l'usuari amb les tecnologies emprades en el disseny, l'àmbit d'ús o l'objectiu, la tria de l'emulador serà diferent. Altres factors com la comunitat que dóna suport a l'emulador o la freqüència de les actualitzacions són també rellevants.

En els següents punts parlarem d'una llista variada d'emuladors disponibles en la actualitat, intentant definir el seu estat, disseny i les funcionalitats més importants.

3.1. GNS3

GNS3 [12] és un emulador inicialment utilitzat per crear i provar xarxes basades en equips Cisco, tot i que avui en dia s'utilitza per provar imatges de diversos fabricants i open-source. Permet emular, configurar i provar des de petites xarxes locals fins a xarxes basades en múltiples servidors i en el cloud. En l'actualitat, GNS3 està desenvolupat i recolzat per una comunitat de 800.000 membres, aproximadament.

Un dels forts d'aquest emulador és la utilitat d'emular imatges de diferents fabricants per tal de provar les compatibilitats entre equips, protocols i solucions. Per aquesta raó, és especialment útil per preparar certificacions Cisco, ja que permet virtualitzar diversos equips del fabricant. Evidentment, aquestes imatges es troben sota llicència de pagament en la majoria dels casos.

3.1.1. Estat actual

GNS3 es troba sota llicència GPLv3. Això vol dir que pot ser utilitzat en entorns comercials, sempre que no es modifiqui el codi per tal de vendre una versió alternativa.

L'última versió és la 2.1, publicada en novembre del 2018 i millorada amb actualitzacions menors amb freqüència mensual. GNS3 compta amb els següents requisits recomanats per fer executar el programa:

- Sistema operatiu: Windows 7 (64 bits) o superior, Mavericks 10.9 o superior, qualsevol distribució de Linux.
- Processador: 2 COREs o més. Extensions de virtualització Intel VT-X o AMD-V.
- Memòria: 4GB RAM.
- Disc: 1GB més l'espai necessari per les imatges a emular.

3.1.2. Arquitectura

GNS3 es pot dividir en quatre parts:

- Interfície gràfica d'usuari (projectes gns3-GUI o gns3-web): mostra una topologia que representa un projecte, i permet interactuar enviant peticions d'API REST HTTP al controlador.
- El controlador (projecte gns3-server): gestiona el cicle de vida d'un projecte. Per millorar les prestacions d'una instal·lació all-in-one, es recomana executar el controlador en una VM a part.
- El computador (part del projecte gns3-server): encarregat de gestionar i executar els diferents emuladors per allotjar les imatges.
- Els emuladors: GNS3 integra amb diferents eines de virtualització i emulació per executar imatges d'equips o màquines virtuals. GNS3 utilitza un mecanisme propi (uBridge) basat en túnels UDP per connectar els nodes de diferents emuladors.

3.1.2.1. Emuladors compatibles

- Dynamips. GNS3 utilitza Dynamips per emular routers Cisco. Es pot exportar una imatge IOS d'un equip físic y utilitzar-la a GNS3. Dynamips és una tecnologia antiga, i no està destinada a versions més noves de IOS (fins a 12.X).
- QEMU. La majoria dels fabricants disposen d'imatges QEMU, que també es poden utilitzar amb GNS3. Per utilitzar imatges de Cisco (IOSv, IOSvL2, IOS-XRv, ASAv), és necessària una subscripció VIRL a Cisco. Les imatges VIRL es creen específicament per emulació, i CISCO dóna suport i les actualitza constantment.
- VMWare/VirtualBox. VMware i VirtualBox ofereixen una manera fàcil de virtualitzar dispositius, i es poden afegir a les topologies GNS3. GNS3 només s'integra amb VMware i VirtualBox, i no té control de la configuració de les màquines virtuals allotjades.
- Virtual PC Simulator (VPCS). VPCS és l'eina lleugera utilitzada per GNS3 per simular equips finals d'usuari. Només permet fer proves de connectivitat.
- Docker. GNS3 ofereix l'opció d'executar diversos contenidors com a part de les topologies. Docker és una bona eina per emular un servidor o un PC que ofereix un servei específic, i es pot utilitzar com un potent reemplaçament de VPCS.

3.1.3. Funcionalitats clau

GNS3 és especialment útil en els següents casos:

- Proves de concepte sense necessitat de maquinari de xarxa, i reduint el temps necessari de desplegament en producció.

- Executar sistemes operatius sense modificar de més de 20 fabricants d'equips de xarxes diferents, en un entorn virtual sense risc.
- Mapes de xarxa dinàmics per a la resolució de problemes.
- Connexió a xarxes físiques: ampliació d'un laboratori connectant directament les topologies GNS3.
- Utilitzar les topologies i laboratoris personalitzats dins de GNS3 per l'àmbit docent.

Més que un emulador de xarxa, GNS3 es pot definir com una eina d'integració d'emuladors de xarxa, que proposa un entorn de treball per construir xarxes basades en diferents fabricants, emuladors i tecnologies de virtualització. GNS3 explota les possibilitats de les diferents màquines virtuals i equips emulats en la topologia de xarxa construïda:

- Suport per múltiples opcions de switching Cisco (ESW16 Etherswitch, IOU/IOL Layer 2 images, VIRL IOSvL2) i suport per OVS amb Docker.
- Suport per totes les imatges VIRL (IOSv, IOSvL2, IOS-XRv, CSR1000v, NX-OSv, ASAv).
- Suport per entorns multifabricant.
- Suport per múltiples hipervisors (Virtualbox, VMware workstation, VMware player, ESXi, Fusion).

Es evident que la principal utilitat de GNS3 és provar imatges CISCO i la seva integració amb altres equips. Tot i així, GNS3 disposa d'un directori d'imatges open-source.

3.2. IMUNES

IMUNES [13] és un emulador/simulador de xarxes multiprotocol basades en IP, desenvolupat per la Universitat de Zagreb. Els nodes virtuals a IMUNES són múltiples instàncies de pila de xarxa que es formen a través de modificacions especials del nucli de FreeBSD.

IMUNES és un emulador de xarxa IP eficient i ràpid, que pot executar centenars de nodes de xarxa virtuals en una màquina física. Cada node virtual té la seva pròpia còpia de la pila de xarxa, així com un sistema de fitxers i espai de processos propis per executar aplicacions. Els nodes virtuals poden connectar amb altres nodes virtuals o amb una interfície de xarxa física, mitjançant enllaços virtuals.

3.2.1. Estat actual

IMUNES està publicat sota llicència BSD, i és propietat intel·lectual de la Universitat de Zagreb, Croàcia. La llicència permet l'ús d'IMUNES i tots els seus components amb fins privats.

IMUNES es va desenvolupar a la Facultat d'Enginyeria Electrònica i Informàtica. L'última versió (2.3.0) va ser publicada al maig del 2019 al directori de GitHub, compatible amb FreeBSD 8 i superiors. En altres sistemes operatius, tals com Linux o Windows, es possible dissenyar gràficament topologies, però no es poden emular els experiments. També es publica una imatge de VMware que conté el sistema operatiu FreeBSD, amb una instal·lació completa i operativa d'IMUNES.

3.2.2. Arquitectura

IMUNES proposa un model d'emulació de xarxes basat en nodes i enllaços virtuals independents, que poden ser configurats i interconnectats individualment.

3.2.2.1. Nodes virtuals

Un node virtual a IMUNES és el conjunt format per una pila de xarxa i una sèrie de processos a l'espai d'usuari, i està basat en vimage de FreeBSD. Cada node és independent de la resta, per la qual cosa manté les seves variables: interfícies de xarxa, taules de routing, comptadors de tràfic, etc. IMUNES integra diverses aplicacions UNIX (dimonis de routing, generador de tràfic, etc.) que poden ser executades als nodes virtuals.

3.2.2.2. Xarxes i enllaços virtuals

Degut a la propietat dels nodes virtuals de ser independents, és necessari un mecanisme de comunicació entre piles de xarxa aïllades. IMUNES fa ús del sistema Netgraph de FreeBSD. Concretament, fa ús dels següents nodes de Netgraph:

- Ng_bridge: implementa connexions de nivell 2 sobre un o més enllaços, i incorpora mecanismes per a la detecció de bucles.
- Ng_eiface: implementa una interfície virtual Ethernet, visible al executar "ifconfig".
- Ng_ether: permet a les interfícies Ethernet interactuar amb el sistema Netgraph.
- Ng_hub: proporciona un mecanisme simple per reenviar els paquets rebuts per un hook per la resta del hooks.
- Ng_pipe: utilitzat pels links Ethernet, permet implementar retard, restriccions d'ample de banda, i altres propietats de l'enllaç.
- Ng_socket: socket a BSD que permet als processos d'usuari participar en el nucli amb Netgraph.
- Ng_wlan: implementa LANs sense fils.

3.2.2.3. Capa de gestió

IMUNES disposa d'un CLI per gestionar les topologies: creació de nodes, assignació d'interfícies físiques o virtuals, i introducció de comandes UNIX per gestió d'aplicacions.

Per una gestió més avançada es recomana la GUI basada en Tcl/Tk, una eina que permet la creació d'interfícies gràfiques a través del llenguatge de script Tcl i la biblioteca Tk.

3.2.3. Funcionalitats clau

IMUNES disposa de dos interfícies d'usuari, la GUI i el CLI. A través del CLI, és possible llançar comandes i gestionar nodes jails prèviament creats. Així, l'emulador està més enfocat en l'ús de la GUI, des d'on podrem simular els següents objectes:

- Link: l'eina per enllaçar els nodes.
- Nodes de capa d'enllaç.
    - Hub
    - LAN Switch
    - External interface: permet connectar un node virtual amb una interfície física.
    - RSTP Switch
    - Filter node: permet filtrar i commutar paquets d'acord al contingut d'aquests.
    - Packet generator: genera i transmet trames de nivell dos a una tassa especificada.

  Els nodes de capa d'enllaç són compatibles amb IEEE 802.1Q, i permeten definir dos models de cues:

    - FIFO (First In First out): els paquets es reenvien en el mateix ordre d'arribada.
    - WFQ (Weighted fair queueing): en funció de certs paràmetres de la capçalera, els paquets es divideixen en diferents cues FIFO, les quals es reparteixen equitativament l'ample de banda disponible.

- Nodes de xarxa.

    - External connection: permet connectar els nodes virtuals al host d'emulació.
    - Router. Utilitza Quagga.
    - Host: representa un servidor o un pc final i no proporciona encaminament. Permet definir rutes estàtiques i serveis de xarxa amb inetd.
    - PC. Només permet definir una ruta per defecte.
    - NAT64: router capaç de traduir entre IPv4 i IPv6 a través de NAT.

Quan els nodes estan connectats a través d'enllaços, ambdós reben uns paràmetres automàticament preconfigurats. Alguns dels paràmetres poden ser visibles: noms d'interfícies, adreces IPv4/IPv6 d'elements de capa de xarxa (PC, host, router), noms de nodes (router1, host1, switch1, pc1, pc2) i etiquetes d'enllaç (ample de banda, retard, BER,...).

L'eina vlink s'utilitza per canviar els paràmetres d'enllaç. Els paràmetres d'enllaç disponibles són els següents:

- Ample de banda (bps).
- Taxa d'error de bits, BER.
- Retard (microsegons).
- Duplicació de paquets (%).

3.3. CORE

CORE [14] (Common Open Research Emulator) és un emulador de xarxes IP que va néixer com un projecte personalitzat d'IMUNES, per part del equip de recerca de xarxa de Boeing Research and Technology.

CORE proposa un model amigable per la creació de xarxes IP, amb la possibilitat de connexió a xarxes físiques. Consta d'una GUI per dibuixar les topologies i executar els experiments. CORE proporciona un entorn per executar aplicacions i protocols basat en virtualització lleugera (contenidors per Linux i jails per FreeBSD).

El projecte CORE defineix una sèrie d'objectius i no objectius per conèixer millor el seu propòsit:

- Amigable: disposa d'una GUI fàcil d'utilitzar.
- Eficient i escalable: LXC i jails permeten nodes més lleugers que les màquines virtuals tradicionals.
- Obert: el projecte CORE insta a la comunitat a proposar millores.
- Útil: CORE està enfocat en oferir una manera real d'estendre xarxes físiques.
- No reinventar la roda: CORE utilitza diverses eines existents per funcionar (Quagga, Linux Bridge, Netgraph, etc.).

3.3.1. Estat actual

Boeing va publicar CORE sota llicència BSD. La versió 5.2.1 de CORE va ser publicada en març de 2019, sota els següents requisits:

- Linux o FreeBSD.
- Processador x86 de 2,0 GHz o superior.
- 2 GB o més de RAM.

- X11 per la GUI.

En la actualitat, el software es pot trobar a GitHub, ja que la pàgina oficial no incorpora les versions més recents. La documentació oficial es amplia, però no inclou informació de les actualitzacions menors, les quals són freqüents.

3.3.2. Arquitectura

El dimoni CORE és l'encarregat de les sessions d'emulació. Construeix les xarxes emulades utilitzant virtualització a nivell del SO per els nodes virtuals, basada en contenidors. CORE funciona amb sistemes Linux i FreeBSD:

- A Linux, CORE utilitza Linux network namespace (també conegut com LXC o Linux Containers) per construir nodes virtuals, i els connecta entre si a través de veth i Linux Bridge al host principal.
- A FreeBSD, CORE utilitza jails com a opció de virtualització de pila de xarxa per crear nodes virtuals, i utilitza diferents nodes del sistema Netgraph de FreeBSD per crear i personalitzar la connectivitat.

El dimoni CORE és controlat a través de la GUI basada en Tcl/Tk, i invoca diferents mòduls de Pyhton per emular els escenaris. La GUI i el dimoni es comuniquen a través de l'API de CORE, basada en sockets. Tot i estar enfocat en la utilització a través de la GUI, CORE inclou una sèrie de comandes bàsiques pròpies per emular els escenaris a través de CLI en Linux (vcmd), mentre que es gestiona amb les comandes habituals de vimage i Netgraph a FreeBSD.

3.3.3. Funcionalitats clau

CORE defineix el següent flux de treball:

1. Definició dels escenaris a través de la GUI.
2. Configuració dels serveis a utilitzar en cada node i dels diferents paràmetres dels enllaços.
3. Execució.

CORE disposa de diferents objectes per construir les topologies desitjades:

- Link. L'eina d'enllaç connecta els diferents nodes, i permet configuració de paràmetres d'enllaç: retard, restriccions d'ample de banda, tassa de pèrdues i duplicat de paquets.
- Nodes de xarxa. Inclouen els següents serveis preestablerts, tot i que es poden modificar.

    - Router: utilitza Quagga.
    - Host. Emula un servidor amb una ruta per defecte i un servidor SSH.
    - PC. Emula un pc final amb una ruta per defecte.

  - Router MDR. OSPF MANET (Mobile Ad Hoc Network) Designated Routers (OSPF-MDR) és una extensió de Quagga OSPF per xarxes ad hoc mòbils, desenvolupada per Boeing. OSPF-MDR optimitza els procés de flooding de LSAs (Link State Advertisment) i millora l'escalabilitat a través de canvis en la formació d'adjacències.

  En l'actualitat la llista de serveis disponibles per nodes de xarxa inclou Quagga, OVS, controlador Ryu, Iptables, nat, vpn, IPsec, ssh, ftp, http i dhcp.

- Nodes de capa d'enllaç.

  - Hub.
  - Switch.
  - Wireless LAN.
  - RJ45. Permet connectar a xarxes físiques a través del host d'emulació.
  - Tunnel. Permet connectar diferents escenaris CORE, localitzats en hosts diferents. Estableix un túnel GRE amb la IP del host destí (Linux o FreeBSD).

3.3.3.1. Xarxes sense fils

El node Wireless LAN permet construir xarxes sense fils. CORE disposa de dos models de xarxes sense fils, que representen diferents graus de precisió:

- Basic on/off. Representa el model menys precís, i utilitza Linux Bridge o ng_Wlan (versió modificada del mòdul ng_hub de Netgraph) en FreeBSD.
- EMANE plug-in. Extendable Mobile Ad-hoc Network Emulator (EMANE) es un framework per modelatge en temps-real de xarxes mòbils. Està desenvolupat per U.S. Naval Research Labs (NRL). Només es troba disponible en Linux.

3.4. Mininet

Mininet [15] és un sistema d'emulació i orquestració de xarxes basades en hosts virtuals, switchs i controladors. Mininet permet la creació de switchs compatibles amb OpenFlow, sent així especialment útil per crear i provar controladors de xarxa SDN compatibles amb OpenFlow.

Per tal de crear xarxes SDN, Mininet utilitza les capacitats de virtualització del nucli de Linux (virtualització lleugera), dividint el sistema en petits contenidors gràcies a la seva pròpia implementació de LXC.

3.4.1. Estat actual

Els primers prototips de Mininet van ser desenvolupats per Bob Lantz i Brandon Heller, investigadors de la Universitat de Stanford. Sota llicència BSD, Mininet està activament desenvolupat per una amplia comunitat, convertint-se així en una plataforma de xarxa virtual molt útil i flexible per a la investigació, l'experimentació i l'aprenentatge.

Mininet 2.2.2 va ser publicat en Març del 2017. La instal·lació recomanada per la comunitat és en forma de màquina virtual; l'arxiu OVF inclou Ubuntu 14.04, Mininet i altres eines bàsiques. La màquina virtual funciona a Windows, Mac i Linux, a través de VMware, VirtualBox, QEMU i KVM.

3.4.2. Arquitectura

Mininet crea xarxes virtuals mitjançant network namespaces, una tecnologia de virtualització lleugera de Linux que, com hem vist anteriorment, proporciona elements de xarxa individuals per cada host. Les xarxes Mininet emulen els següents components:

- Enllaços. Utilitzant veths, Mininet crea túnels per connectar els diferents elements desitjats de la topologia.
- Hosts. Conjunt de processos a nivell d'usuari associats a un espai de noms de xarxa. Qualsevol aplicació compatible amb Linux es pot executar en un host Mininet.
- Switchs. Hi ha diferents opcions, tals com Linux Bridge o OVS.
- Controladors. Mininet integra amb diferents controladors OpenFlow (OVS, NOX, RYU...).

Per defecte, cada host és emulat en un espai de noms, mentre que els switchs i els controladors es troben al espai de noms root, és a dir, al host d'emulació. Aquest comportament és clau per entendre la connectivitat entre els switchs i els controladors, proporcionada des del propi host d'emulació. Aquest comportament es pot modificar amb l'opció *innamespace*.

3.4.3. Funcionalitats clau

Mininet inclou un CLI per gestionar tota la xarxa, així com la possibilitat d'invocar el CLI d'un node en concret. El CLI de Mininet permet diferents opcions a través de la comanda *mn* per crear experiments. L'API en Python de Mininet és una altre eina, més potent, per crear i executar experiments, i disposa d'alguns scripts d'exemple per començar a interactuar. Mininet permet importar llibreries per suportar topologies amb diferents paràmetres. Per exemple, és possible configurar un límit de CPU per un host. Els links també son configurables en aquest sentit: ample de banda, retard, mida màxima de la cua en paquets i pèrdues.

Mininet incorpora tres tipus de topologies preconfigurades:

- Single: un switch connectat a tots els hosts.
- Linear: switchs en cascada interconnectats amb un host per cadascun.
- Tree: en forma d'arbre, amb longitud i profunditat configurable.

L'API de Mininet està documentada, i conté múltiples classes per realitzar tasques tan senzilles com iniciar una topologia amb un switch, o altres més complexes com connectar les topologies a xarxes externes a través de la classe NAT. El principal fort de l'emulador és la implementació amigable que fa de totes les tecnologies de virtualització que utilitza, facilitant a l'usuari una sèrie de mètodes en Python que permeten elaborar les topologies sense coneixements de les diferents eines involucrades en el procés.

Una de les funcions més útils de Mininet és la compatibilitat amb SDN. Mitjançant el protocol OpenFlow, és possible programar els switchs per realitzar qualsevol tasca amb els paquets.

3.5. Cloonix

Cloonix [16] és un emulador programat en C per crear xarxes de màquines virtuals basades en KVM. Cloonix funciona com a eina per automatitzar la creació i configuració de les màquines virtuals QEMU-KVM, per posteriorment enllaçar-les a través de cables LAN emulats, basats en sockets. Seguint una arquitectura servidor-client, Cloonix proporciona accés a les màquines virtuals emulades a través d'un CLI. Cloonix es defineix com una eina útil per fer demostracions, test regressius i proves de software de xarxa sense impactar el sistema amfitrió.

3.5.1. Estat actual

Cloonix és distribuït sota la llicència AGPLv3. L'equip de Cloonix actualitza el software sovint amb versions experimentals (no estables) a GitHub, tot i que l'emulador no compta amb una gran comunitat activa d'usuaris.

La versió Cloonix 02-03 va ser publicada en maig de 2019, compatible amb distribucions Linux.

3.5.2. Arquitectura

Cloonix consta de dos parts principals: servidor i client. El servidor crea i gestiona els diferents objectes que Cloonix és capaç d'emular, mentre que el client proporciona el control a distància i la visualització de tots aquests objectes. Els objectes representen els nodes i les diferents funcionalitats que Cloonix és capaç d'emular.

Una sola màquina física pot executar, teòricament, un nombre il·limitat de servidors, especificant nom, IP i port. Cada servidor pot gestionar 40 nodes, i un nombre il·limitat de la resta d'objectes. Per tal d'interactuar amb els escenaris, CORE disposa de diferents clients. Cada client especifica el nom, la IP, el port i una clau per connectar a un servidor. Alguns dels clients més importants són:

- Cloonix_zor. Es tracta d'una GUI que permet accedir a les topologies de tots els servidors. Està basada en gtk3, un conjunt d'eines per desenvolupar interfícies gràfiques.
- Cloonix_cli. CLI per accedir a les diferents funcions de l'emulador per cada servidor. Eina clau per afegir, configurar o eliminar objectes d'una topologia associada a un client.
- Cloonix_ssh. Permet llançar comandes a la CLI d'una màquina virtual.
- Cloonix_ice. Proporciona accés al escriptori remot d'un màquina virtual a través del projecte Spice.
- Cloonix_scp. Permet accedir al sistema de fitxers d'una màquina virtual.

3.5.3. Funcionalitats clau

Cloonix és capaç d'emular els següents objectes a través de la GUI o el CLI:

- Lan. Emula un cable entre màquines virtuals del mateix servidor. Qualsevol paquet enviat serà rebut per totes les màquines virtuals. Cloonix no especifica ni defineix la categoria del objecte (hub, switch, router, etc.), tot i que actua com un hub. Això comporta certes ineficiències, i l'equip de Cloonix ha anunciat (Juny 2019) una versió experimental amb OVS i la llibreria DPDK, per millorar el processament de paquets entre màquines virtuals.
- KVM. Instància de màquina virtual QEMU-KVM. Cloonix disposa de diferents maneres (escriptori remot amb Spice o X11, tmux, ttySO serial, protocol scp) per connectar a les màquines virtuals.
- Tap. Permet accés al host d'emulació des de les màquines Cloonix, formant una connexió amb una interfície del host.
- C2C. Cloonix permet fins a 40 màquines virtuals gestionades per un sol servidor. Per tal de crear xarxes més grans, és necessari un mecanisme de connexió entre màquines de diferents servidors. Aquest objecte s'anomena c2c.
- SNF. Eina per capturar i analitzar paquets. Crea un arxiu .pcap que pot ser obert amb eines com Wireshark.
- NAT. Permet accés a xarxes externes des de les màquines virtuals.
- A2B. Eina configurable per crear retard o pèrdues de paquets entre màquines virtuals.

3.6. Virtual Networks over Linux (VNX)

VNX [17] és una eina de virtualització dissenyada amb el propòsit d'ajudar a construir bancs de proves de forma automàtica. Permet definir i desplegar escenaris de xarxa amb nodes basats en contenidors o màquines virtuals de diferent tipus, interconnectats a través d'una topologia definida per l'usuari en XML.

VNX es basa en l'eina d'emulació de xarxes VNUML, que es va desenvolupar l'any 2003 en el context de la participació del Departament d'Enginyeria de Sistemes de Telemàtica de la Universitat Politècnica de Madrid en el projecte de recerca Euro6IX. VNX és una evolució per millorar les limitacions de VNUML, tals com oferir un entorn capaç d'emular nodes amb diferents tecnologies de virtualització i la possibilitat de gestió individual d'aquests nodes.

3.6.1. Estat actual

L'actual versió de VNX és la 2.0b, sota llicència GNU GPL. L'equip del DIT-UPM encarregat del desenvolupament de VNX està coordinat pel professor David Fernández.

VNX no compta amb una gran comunitat activa, i les actualitzacions del software són esporàdiques. La pàgina web oficial del projecte, on podem trobar tota la documentació i alguns exemples d'ús, ha estat indisponible en algunes ocasions durant la realització d'aquest treball.

Els requisits per executar l'última versió són:

- Distribució de Linux recent.
- Processador amb suport per extensions de virtualització (en cas de màquines virtuals KVM).
- 2GB RAM.
- 10GB de disc.

VNX disposa d'una màquina virtual OVA preparada per fer córrer l'emulador amb Virtual Box, tot i que aquesta només suporta nodes amb LXC.

3.6.2. Arquitectura

VNX segueix una arquitectura basada en mòduls. Els components de VNX es poden separar en tres blocs. El bloc principal està format pel mòdul central de VNX. Aquest mòdul es l'eix de VNX, i representa la interfície entre l'usuari i l'emulador. Durant el cicle d'execució, aquest mòdul invoca i orquestra els diferents mòduls i fitxers necessaris per implementar les diferents opcions de VNX. Aquests mòduls de suport representen el segon bloc de VNX. Finalment, el mòdul principal inicialitza els mòduls de l'API de virtualització; aquest mòduls són plugins que doten a VNX de la capacitat de donar suport a diferents

plataformes de virtualització de nodes (Dynamips, UML, LXC, KVM, XEN, Windows, entre altres). L'API ofereix una interfície on el mòdul principal llança les diferents peticions, per a que cada plugin sigui responsable de la traducció d'aquestes peticions i d'operar cada tecnologia de virtualització.

L'API de virtualització de VNX és una versió simplificada de l'API de libvirt, col•lecció de software que proporciona una gestió unificada de màquines virtuals i contenidors de diferent tipus.

3.6.3. Funcionalitats clau

En quan a l'experiència d'usuari, VNX està basat en dos components:

- Els escenaris virtuals descrits en llenguatge XML.
- Un intèrpret encarregat de construir i gestionar els escenaris de xarxa seguint les instruccions de l'usuari rebudes pel CLI.

Els escenaris es defineixen a través dels diferents tags en el fitxer XML:

- <*global*>. Es defineixen tots els paràmetres generals per defecte de les màquines virtuals (mode d'accés, sistema de fitxers, subxarxa de gestió...), així com paràmetres del sistema VNX. També es defineix l'activació de STP en els switchs creats per VNX, i el mode promiscu per les interfícies, utilitzant el tag <*netconfig*>.
- <*net*>. Informació de les xarxes virtuals utilitzades a l'escenari. Es defineix un mode (OVS, Linux Bridge o UML switch) i un tipus (LAN o PPP). En el cas de UML switch, permet configurar filtres per capturar tràfic amb sintaxis tcpdump. Altres paràmetres per OVS i Linux Bridge són:
    - <*externa*>. Permet especificar una interfície del host d'emulació per la connexió a xarxes externes. Només disponible a Linux Bridge.
    - <*VLAN*>. Permet especificar els diferents ID's de les VLAN a utilitzar i mode trunk.
    - <*stp*>. Només disponible a OVS, permet activar Spanning Tree Protocol.
    - <*controller*>. Només disponible a OVS, permet connectar a un controlador OpenFlow.
    - <*of_version*>. Només disponible a OVS, permet especificar la versió OpenFlow a utilitzar.
- <*vm*>. Paràmetres específics de la màquina virtual al escenari, tals com les interfícies o els recursos assignats. Alguns dels més destacats són:
    - <*type*>. Libvirt (per màquines KVM o xen, entre altres), Dynamips, UML o LXC.

- <*filesystem*>. S'indica el directori al fitxer de sistema que es carregarà al node. VNX proporciona models de fitxers de sistema per diferents nodes.
- <*if*>. Aquest tag permet incloure configuració d'una interfície tal com id, xarxa a la qual connecta, tag de VLAN i direcció IP.
- <*route*>. S'especifica un ruta estàtica per defecte.
- <*forwarding*>. Permet activar el reenviament de paquets.

- <*host*>. Paràmetres d'un host que participa en l'escenari, similars als de <*vm*>.

Una vegada definit l'escenari, VNX disposa de dos modes d'operació diferents:

- Mode de creació d'escenaris: s'executa i es finalitza un experiment a partir d'una topologia definida.
- Mode d'execució de comandes: ens permet llançar comandes a les màquines virtuals des de la consola del mòdul principal de VNX.

3.7. Conclusions

Taula 3.1. Comparativa dels emuladors de xarxa

	GNS3	IMUNES	CORE	MININET	CLOONIX	VNX
Llicència	GPLv3	BSD	BSD	BSD	AGPLv3	GNU GPL
GUI/CLI per creació de topologies	Sí/No	Sí/Sí	Sí/Sí	No/Sí	Sí/Sí	No/Sí
Nodes VM	Sí	No	No	No	Sí	Sí
Nodes en contenidors	Sí	Sí	Sí	Sí	No	Sí
Imatges multi-fabricant	Sí	No	No	No	Sí	Si
Enllaços configurables	Sí	Sí	Sí	Sí	Sí	No
L2 switch	Sí	Sí	Sí	Sí	No	Sí
802.1Q VLAN tag	Sí	Sí	No	Sí	No	Sí
STP	Sí	Sí	No	Sí	No	Sí
OpenFlow	Sí	No	Sí	Sí	No	Sí
Connexió a xarxes externes	Sí	Sí	Sí	Sí	Sí	Sí
Inspecció de paquets	Sí	Sí	Sí	Sí	Sí	No
RIP	Sí	Sí	Sí	No	No	No

OSPF	Sí	Sí	Sí	No	No	No
IS-IS	Sí	Sí	Sí	No	No	No
BGP	Sí	Sí	Sí	No	No	No
Instal·lació de nous serveis	Sí	Sí	Sí	Sí	Sí	Sí

Fins ara hem vist emuladors pensats per diferents propòsits. En l'anterior taula recollim una sèrie de característiques importants per valorar-los. És important recalcar la subjectivitat, en cert grau, al qualificar els atributs d'un emulador; tenim en compte la compatibilitat o la integració dels diferents atributs? En la majoria dels casos ens trobem amb emuladors on l'usuari pot instal·lar certs serveis en el nodes, sense que estiguin prèviament integrats en la GUI, l'API o el CLI de l'emulador. Un exemple seria Mininet, on podem instal·lar Quagga en els nodes manualment o desenvolupar el codi adient en un mòdul al controlador. Per tant, hem establert com a criteri la integració de manera nativa dels diferents atributs.

GNS3 és probablement l'emulador més utilitzat i conegut de la llista. És especialment útil, i així esta enfocat, per emular equips Cisco. Concretament, és ideal per construir topologies basades en imatges d'equips reals i nodes amb software lliure, utilitzant contenidors o màquines virtuals. Així, permet estudiar la compatibilitat de protocols propietaris amb múltiples eines open-source. En aquest sentit, VNX també és una opció interessat, ja que permet integrar múltiples tecnologies de virtualització per nodes gràcies a la llibreria Libvirt, així com el suport per Dynamips i LXC. VNX és probablement la millor opció per usuaris familiaritzats amb XML, tot i que la manca de serveis integrats fa que l'usuari sigui responsable de la instal·lació posterior d'aquests. Per altra banda, Cloonix és l'altre emulador compatible amb màquines virtuals QEMU-KVM, però no acaba d'aportar altres beneficis com fan emuladors com GNS3 o VNX.

Mininet és l'emulador ideal per usuaris familiaritzats amb desenvolupament de codi, i que busquen una forma escalable i eficient de crear grans infraestructures de xarxa. Les possibilitats són infinites en aquest sentit, gràcies a l'API i la integració amb controladors SDN, des d'on és possible establir qualsevol comportament en la xarxa. No obstant, no és la millor opció per construir xarxes a petita escala amb protocols de comunicació o gestió de xarxa tradicionals, ja que trobem altres opcions més ràpides en emuladors que integren més eines. També pot ser desaconsellable per usuaris no familiaritzats amb desenvolupament de codi en Python o Java, llenguatges àmpliament utilitzats per controladors SDN.

IMUNES i CORE són dos projectes que proporcionen una manera ràpida i amigable, gràcies a la GUI, de provar infinitat de serveis i protocols, molts d'ells integrats de manera nativa en el cas de CORE. Ambdós basats en contenidors com a tecnologia de virtualització de nodes, utilitzen múltiples eines open-source per realitzar les diferents tasques pròpies d'un emulador. En el cas de CORE, trobem avanços respecte IMUNES, tals com la compatibilitat amb

Linux, les millores en les opcions de la GUI i la integració de nous serveis. També trobem certs retrocessos, com la desaparició del servei integrat per 802.11q.

Per aquest treball, utilitzarem CORE per explorar els diferents casos d'ús d'un emulador. CORE s'ajusta als recursos disponibles, i és teòricament un dels emuladors més complets analitzats. A més, amb la inclusió dels nous serveis i de les noves funcionalitats a través de CLI, representa una alternativa menys coneguda a Mininet per desplegar grans infraestructures de xarxa.

CAPÍTOL 4. Explorant l'emulació de xarxes amb CORE

Per a la instal·lació de l'emulador CORE, utilitzarem una màquina virtual a Virtual Box, i un servidor basat en X11 per poder visualitzar les diferents interfícies gràfiques al host. CORE posa a disposició de la comunitat una màquina virtual a la pàgina oficial, que inclou la versió 4.7 amb Ubuntu 14.04. Al directori GitHub, trobem versions més actuals del codi font de l'emulador, que inclouen certes millores, com la possibilitat d'utilitzar el servei OVS als nodes o la compatibilitat amb versions més noves d'Ubuntu.

Per a la realització del treball, crearem una màquina virtual amb Ubuntu 18.08, 4GB de RAM, 4 processadors lògics (Intel® CORE™ i5-8350U 1,70GHz) i 10GB de disc assignats dinàmicament. Una vegada creada la màquina virtual, es procedeix a la instal·lació de CORE 5.2.1 a partir del codi de GitHub. Una altre opció seria instal·lar CORE a partir dels paquets del directori de software oficial de Linux, però aquesta via no proporciona la versió més recent de l'emulador. Cal recalcar que, instal·lant l'emulador des del codi de GitHub, és necessari fer la instal·lació de les dependències i de tot el software de suport que CORE utilitza per incorporar els diferents serveis. Quagga, Wireshark o IPsec-tools són exemples de les eines que requereixen ser instal·lades per l'usuari juntament amb l'emulador. En aquest sentit, CORE és flexible i deixa a criteri de l'usuari la instal·lació dels serveis que seran utilitzats en cada cas.

Finalitzat el procés d'instal·lació, aixequem el dimoni CORE i la GUI. A l'esquerra trobem classificats els diferents elements que CORE permet emular. A la barra superior trobem les diferents opcions de configuració de la GUI, la gestió del projecte, les eines per interactuar amb l'escenari i els widgets. CORE permet obrir i desar en format IMN o XML. El format IMN prové del emulador IMUNES, i proporciona compatibilitat entre ambdós, mentre que el format XML va ser definit per CORE.

CORE proporciona diferents eines per configurar i provar els entorns, entre les quals destaquen:

- Traffic: permet establir un flux de tràfic UDP o TCP, establint els nodes origen i destí.
- IP/MAC adresses: permet configurar les direccions IP i MAC que CORE assigna als nodes a mesura que es crea la topologia.
- Topology generator: genera certes topologies de routers (estrella, cadena, arbre...).

La secció de widgets permet afegir diverses indicacions útils sobre la topologia, per tal de tenir més informació durant l'execució dels experiments. Observar el procés de formació d'adjacències OSPF i identificar el tipus de veïns OSPF, observar l'ample de banda d'un enllaç en temps real o veure les polítiques de firewall i IPsec, són algunes de les opcions. CORE només permet establir un widget d'observació en cada emulació.

Per tal de gestionar els diferents contenidors LXC, CORE incorpora les seves pròpies eines i no utilitza el paquet lxc-tools. Cada vegada que es crea un

contenidor, CORE llança el dimoni vnoded com a procés. Aquest dimoni escolta en un canal de control comandes per iniciar altres processos al node. El canal de control és un socket de domini Unix localitzat a /tmp/pycore.X/nY, on X i Y representen la sessió i el node. CORE proporciona l'eina vcmd per tal d'interactuar amb el dimoni vnoded d'un contenidor, i poder així llançar comandes al node. Finalment, per controlar les connexions entre contenidors al host principal, CORE utilitza Virtual Ethernet pairs connectats a través de Linux Bridge.

En els següents apartats construirem diversos escenaris que ens permetin explorar les possibilitats de l'emulació de xarxes de la mà de CORE. A través del diferents escenaris, tractarem d'il·lustrar i explicar les capacitats, les limitacions i les particularitats de l'emulador, fent èmfasis en les parts no documentades.

4.1. Desplegament de topologies i configuració de serveis

Al llarg del següent apartat, construirem una topologia basada en serveis integrats en CORE; en concret, utilitzarem Quagga i serveis de forwarding de Linux. Al ser el primer escenari, explicarem detalladament el comportament de tot l'escenari, utilitzant les diferents eines de CORE, per tal de demostrar el correcte comportament de l'emulació de xarxes. L'objectiu és comprovar com és possible emular xarxes a CORE i obtenir el mateix resultat que en xarxes basades en nodes físics, interactuant i observant el comportament de tots els elements involucrats.

En el primer escenari utilitzarem routers per crear una topologia basada en OSPF. Per defecte, a mesura que s'afegeixen routers a l'escenari, CORE assigna IP's i MAC's basant-se en les direccions i pautes configurades prèviament. Utilitzarem el rang IP 10.0.0.0/24 (2001::), i les direccions MAC s'assignaran començant per 00:00:00:aa:00:00.

CORE activa els serveis de OSPFv2 i v3, zebra i IPForward (permet rebre i enviar paquets en interfícies de sistemes Linux) quan es crea una topologia, i assigna tots els routers a l'àrea 0. En el primer escenari, editarem aquesta configuració per utilitzar OSPFv2 i crear una topologia amb tres àrees.

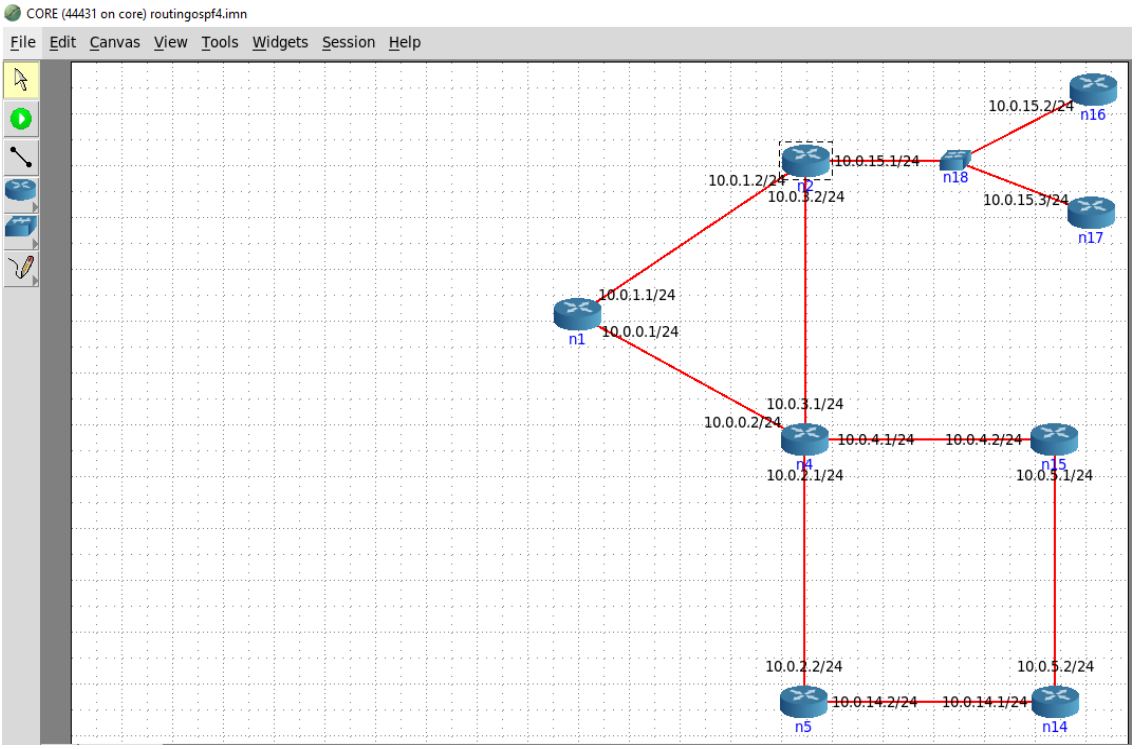

Fig. 4.1 Escenari amb 3 àrees OSPF

Per configurar l'escenari, accedim a la part de serveis dels nodes. Aquí habilitarem els serveis a utilitzar (**Fig. 4.2**); OSPFv2, zebra i IPForward. Un cop habilitats els serveis, és necessària la configuració d'aquests per part de l'usuari. CORE mostra una finestra on s'especifica el fitxer del servei a configurar (el directori dintre del node en concret), i les opcions de configuració mitjançant text o carregant un fitxer (**Fig. 4.3**). En el nostre cas, configurarem zebra per obtenir la topologia desitjada: tres àrees i n2 amb prioritat per ser DR (Designated Router) en totes les interfícies.

- Àrea 1: N4, N15, N5 i N14
- Àrea 0: N4, N1 i N2
- Àrea 2: N2, N16 i N17

Al finalitzar la configuració de tots els nodes, activarem la visualització dels widgets d'adjacència (**Fig. 4.4**) i veïnatge OSPFv2.

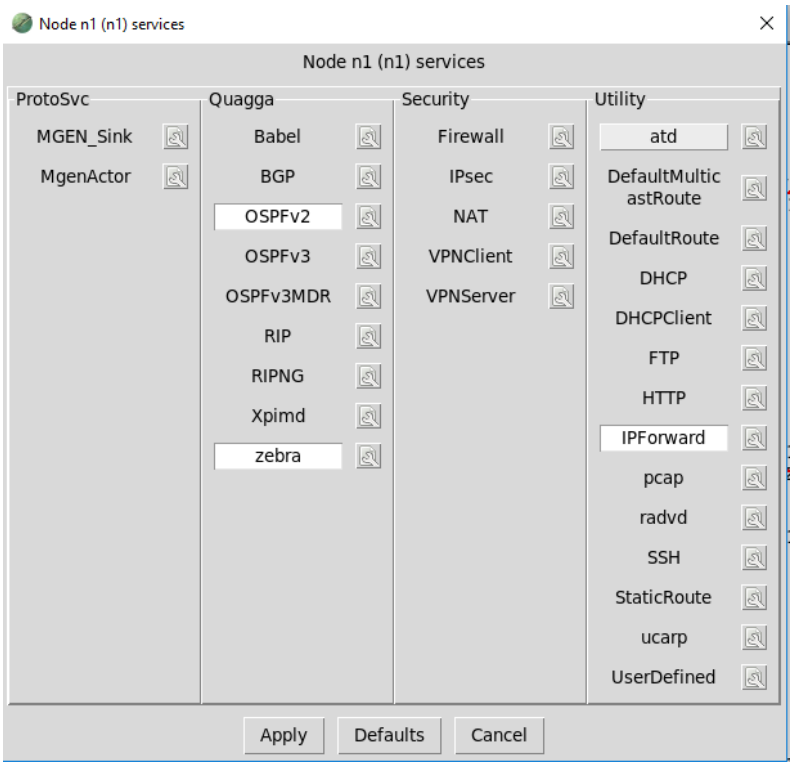

Fig. 4.2 Activació de serveis del primer escenari

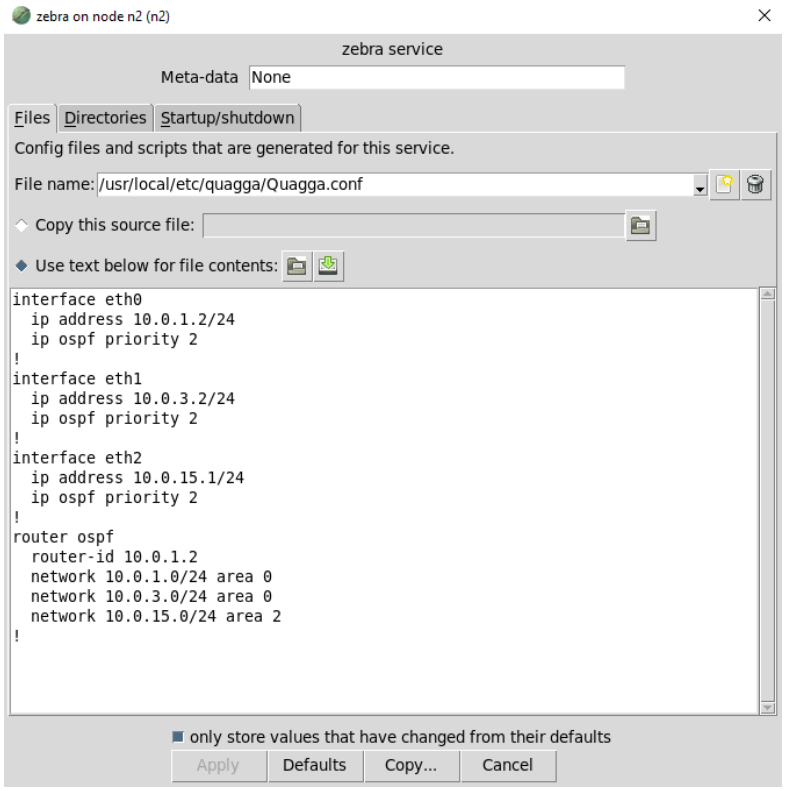

Fig. 4.3 Configuració de serveis: Quagga

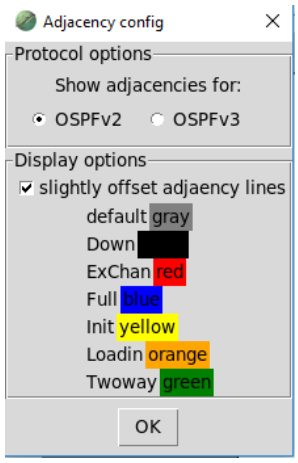

Fig. 4.4 Widget d'adjacència OSPF

En iniciar l'emulació, tenim accés a diferents formes de connexió i anàlisis de paquets en cada node:

- CLI: vtysh (Quagga), sh i bash.
- Anàlisis de paquets: tcpdump, tshark i Wireshark.

Iniciada l'emulació, podem veure en les següents figures la informació proporcionada pels widgets seleccionats. Per exemple, utilitzant N2 i l'àrea 2 com a referència, podem accedir a visualitzar, en Wireshark, la interfície eth2 (10.0.15.1/24).

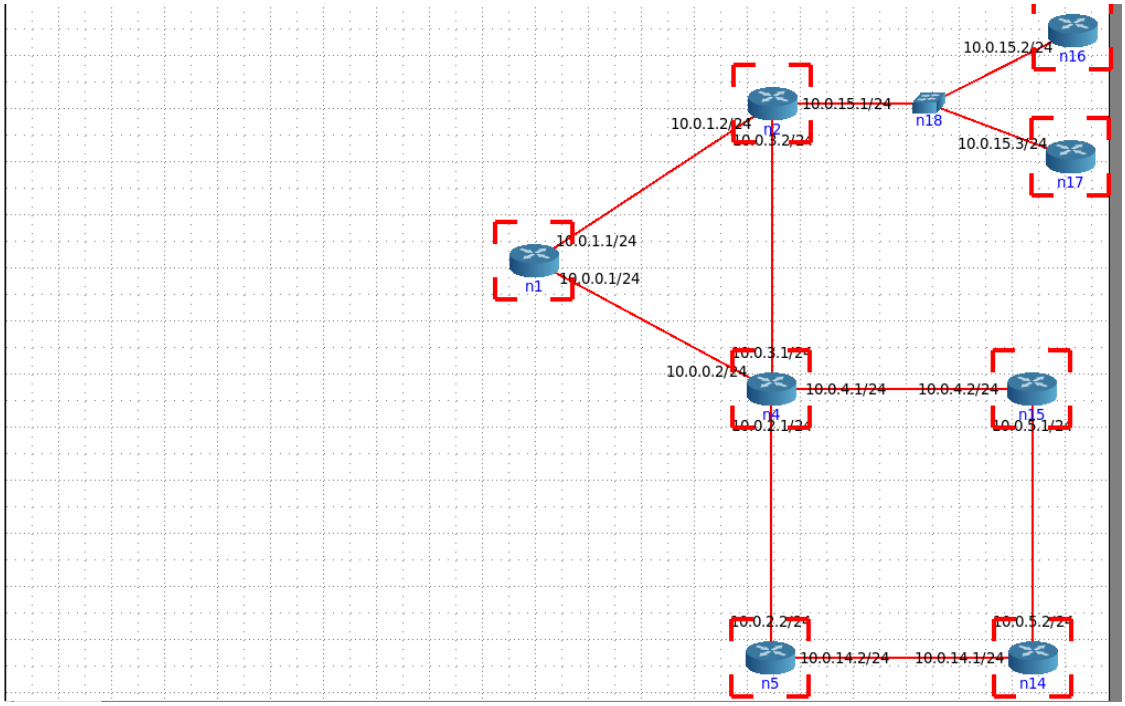

Fig. 4.5 Formació d'adjacència OSPF: exchange

- Elecció DR i BDR (Backup Designated Router): els tres routers del segment (N2, N16 i N17) envien Hello a 224.0.0.5, que representa l'adreça multicast de tots el routers OSPF. Cada router indica la seva prioritat, i finalment N2 és escollit DR (prioritat 2) i N17 BDR, al tenir un router ID més gran que N16.

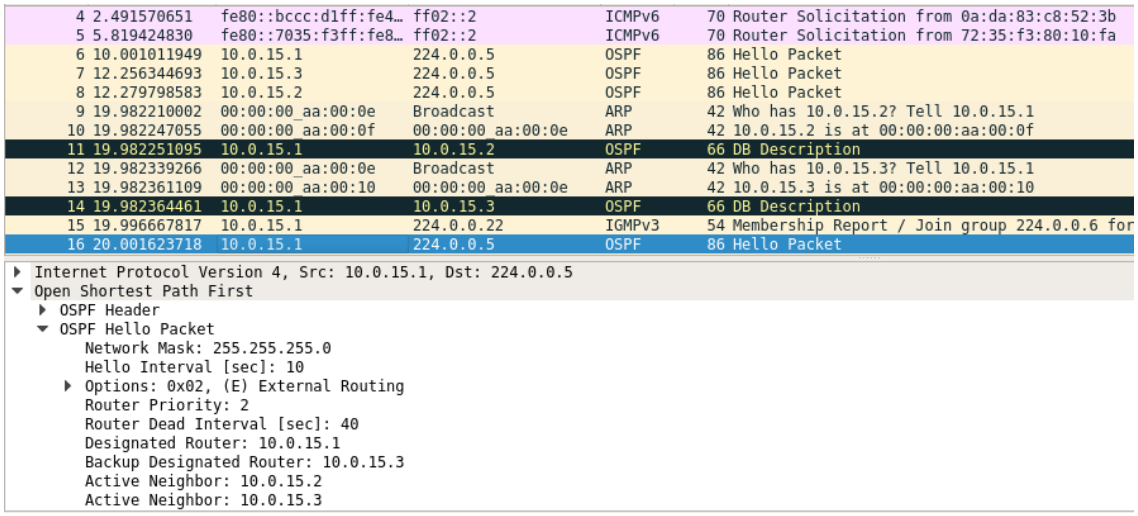

Fig. 4.6 Paquet Hello enviat per N2, amb informació de l'àrea 2

- Formació d'adjacència amb n17:
  - Init: S'envien els paquets de Hello però encara no s'ha establert una connexió bidireccional.

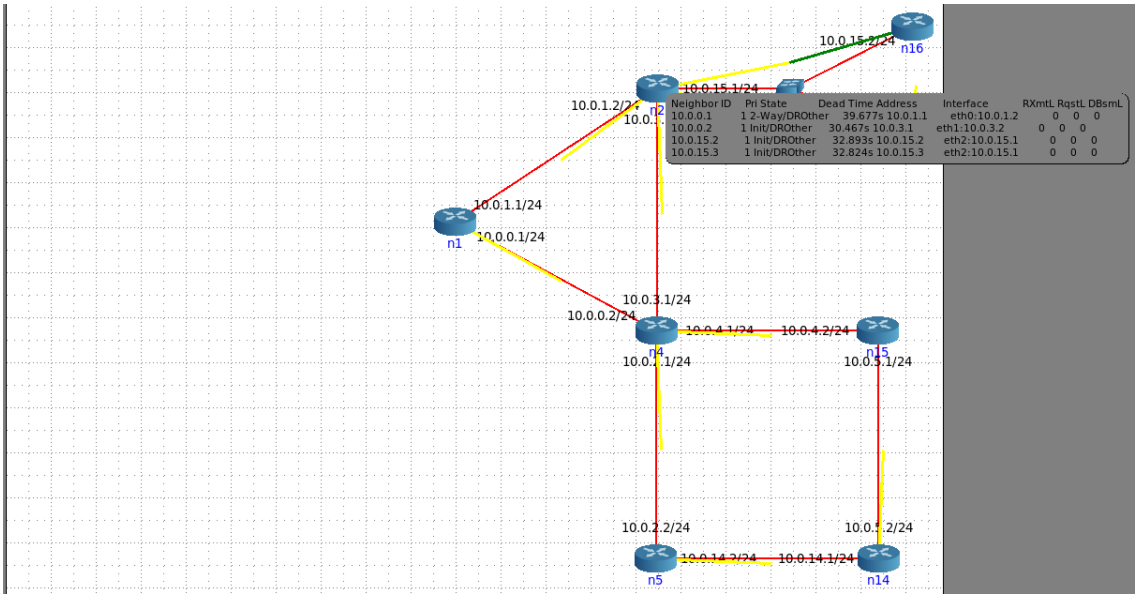

Fig. 4.7 Formació d'adjacència OSPF: init

  - 2-way: Es rep el Hello de tornada amb l'ID propi, i s'estableix connexió bidireccional.

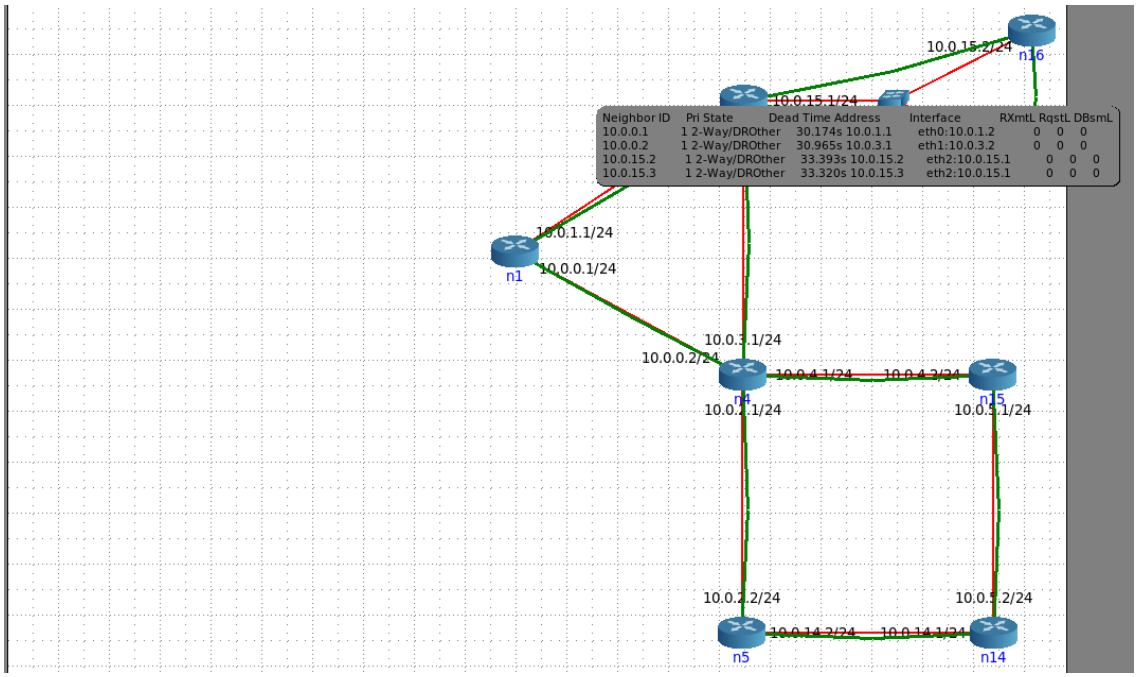

Fig. 4.8 Formació d'adjacència OSPF: 2-way

- Exstart: S'inicia la negociació per determinar el màster, que controlarà l'intercanvi de dades. Tot i que N2 intenta ser màster, N17 és escollit màster degut al ID més gran, 10.0.15.3 (**Fig. 4.9**).

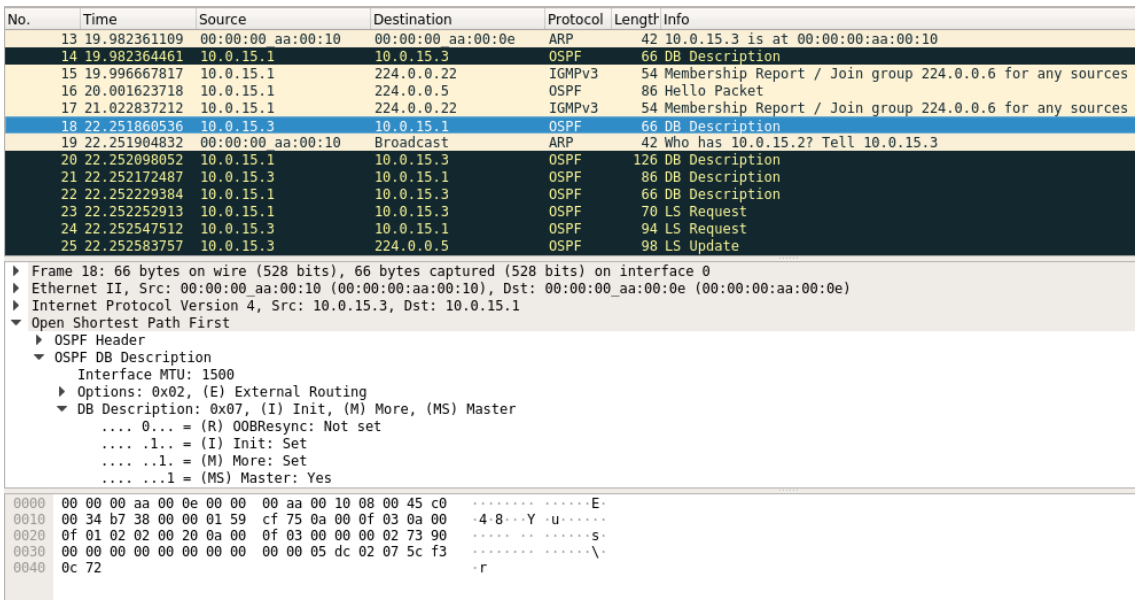

Fig. 4.9 Intercanvi de dades entre N17 i N2

- Exchange: Intercanvi de les bases de dades. N2 respon als paquets DB de N17, amb el mateix numero de seqüència. Si un

router detecta que li falta un LSA (Link State Advertisement) a partir dels DB rebuts, emet un LS Request, que és respost amb un LS Update. Finalment, el router ha de confirmar la rebuda amb un LS Acknowledge.

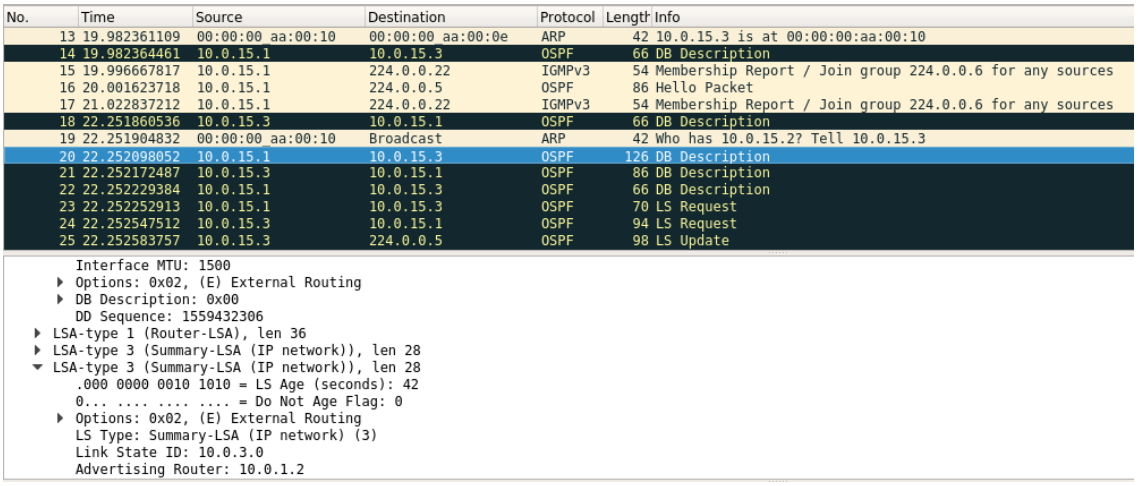

Fig. 4.10 Intercanvi de dades entre N17 i N2

- Loading: aquest estat indica que un router ha finalitzat de transmetre informació, però encara rep del router veí.
- Full: routers completament sincronitzats, l'adjacència està formada (**Fig. 4.11**).

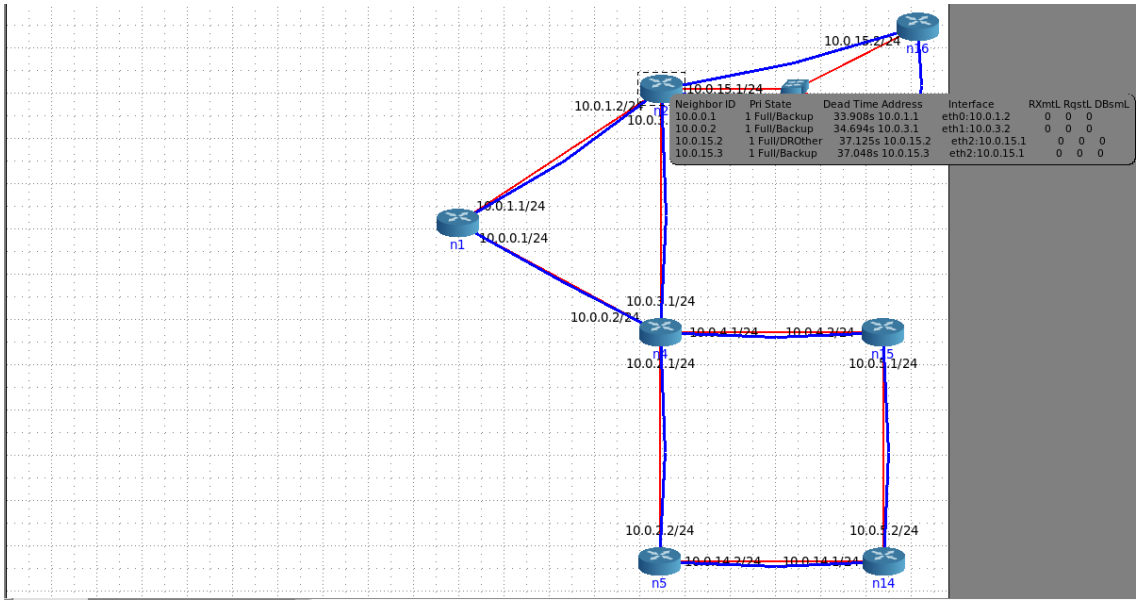

Fig. 4.11 Formació d'adjacència OSPF: full

CORE permet accedir a diferents formes de CLI. Per tal d'interactuar amb l'escenari realitzat, accedim al vtysh. Agafant com a referència N2 i l'àrea 2, podem veure la informació de la base de dades amb "*show IP ospf database*" (**Fig. 4.12**):

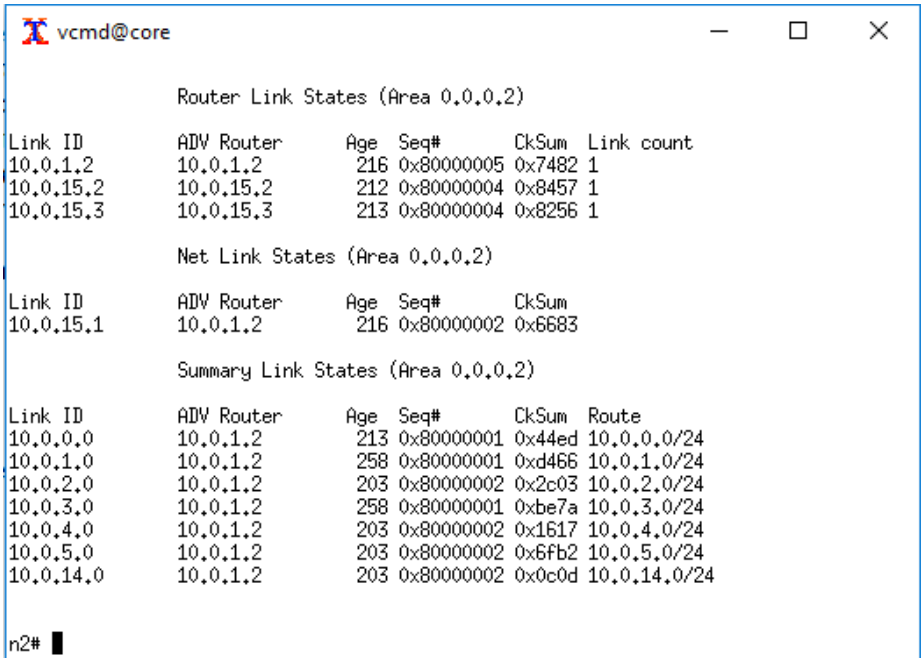

Fig. 4.12 Vtysh N2: show IP ospf database

- Network LSA (**Fig. 4.13**): Creat pel router designat (N2), descriu els routers connectats a la xarxa de trànsit.

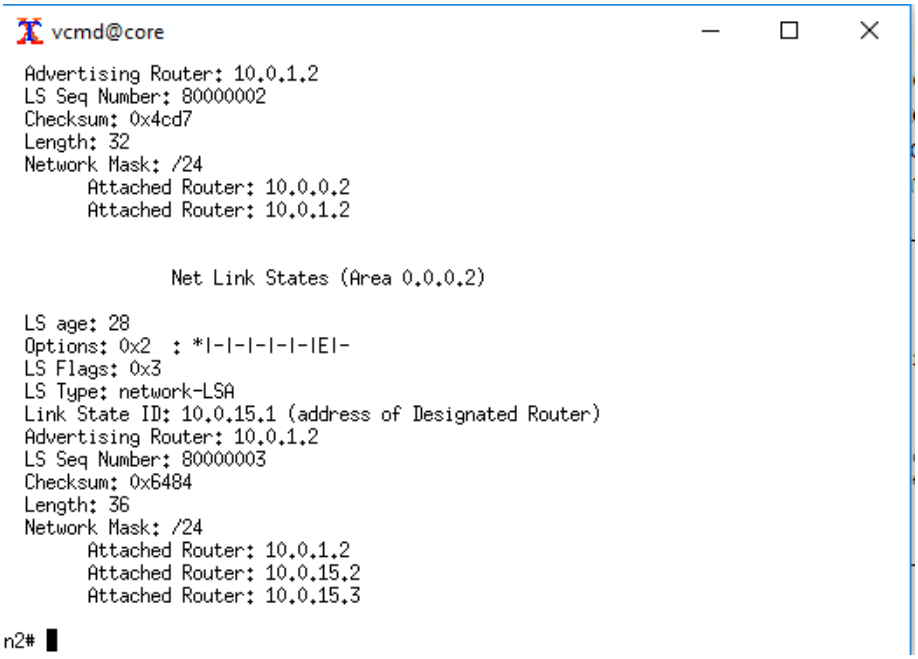

Fig. 4.13 Vtysh N2: show ip ospf database network

- Summary LSA (**Fig. 4.14**): generats per N2 com a ABR (Area Border Router), informen sobre una xarxa fora de l'àrea.

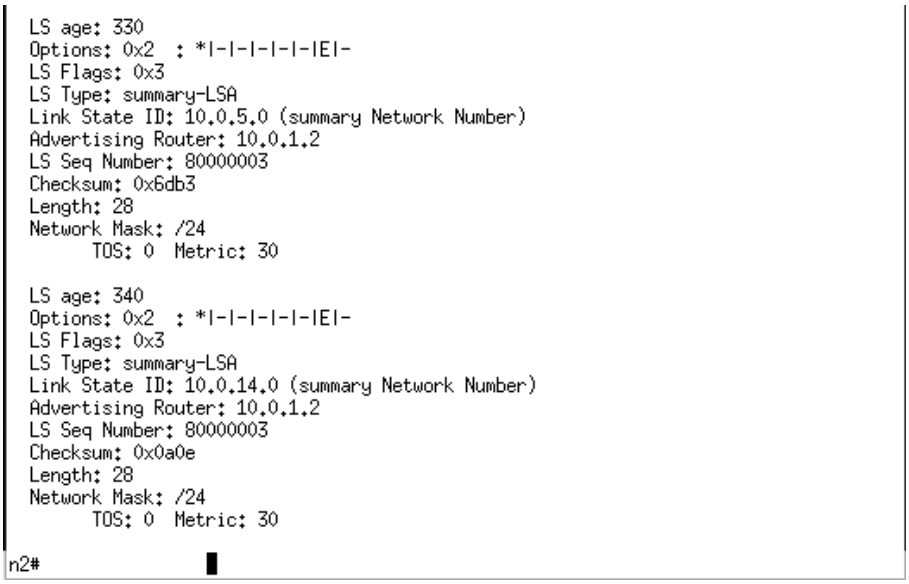

Fig. 4.14 Vtysh N2: show ip ospf database summary

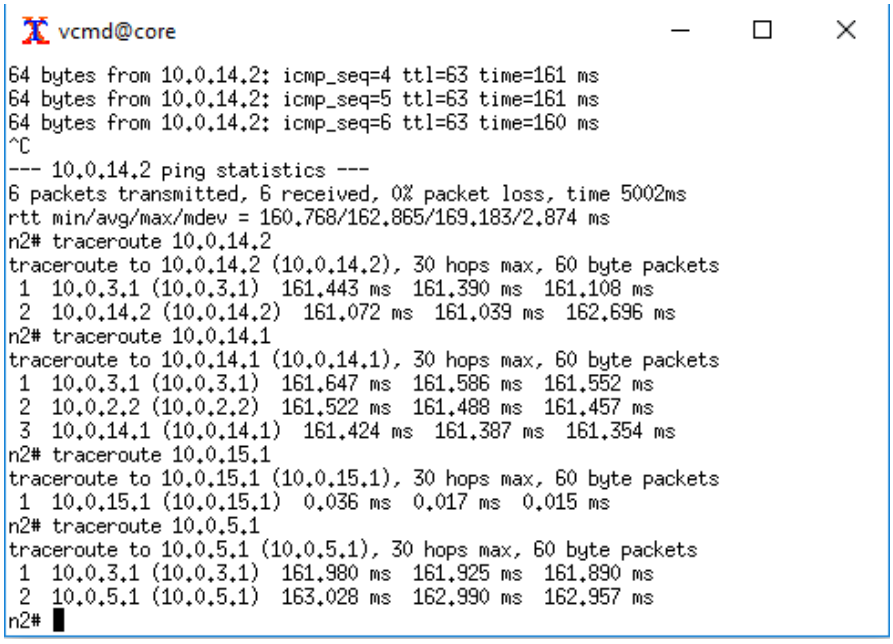

Fig. 4.15 Connectivitat des de N2 a l'àrea 1

A continuació, ampliarem la xarxa amb un altre AS (Autonomous System). A l'esquerra trobarem l'AS 2, i a la dreta l'AS 1. AS 2 utilitzarà també OSPF com a IGP (Interior Gateway Protocol), de forma que tindrem un àrea 0. L'AS 2 tindrà dos routers externs connectant amb l'AS 1 a través de EBGP (External Border Gateway Protocol). N1 advertirà el prefix 10.0.0.0/16, mentre que N3 i N6 advertiran el prefix 172.16.0.0/16.

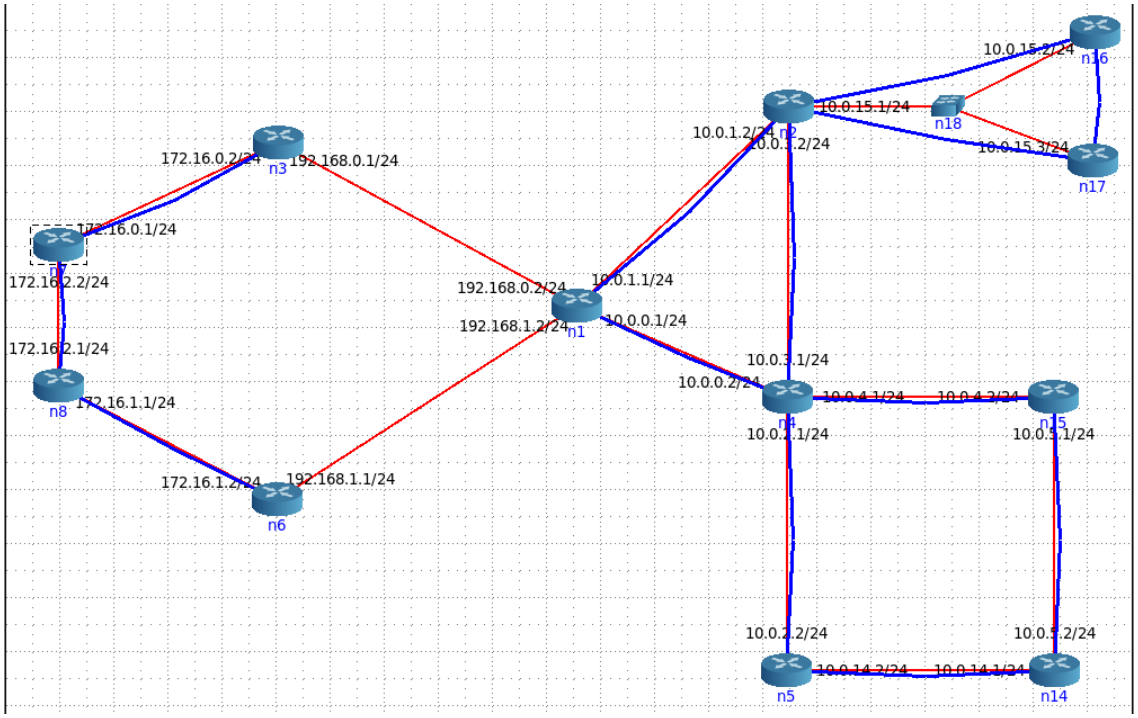

Fig. 4.16 Escenari ampliat: 13 nodes

En BGP no hi ha descobriment automàtic de veïns, i les sessions s'han de configurar manualment, indicant el veí i l'AS. BGP utilitza el protocol TCP com a protocol de transport. Amb l'ajuda de Wireshark, podem visualitzar l'estat de la connexió entre N3 i N6 (IBGP). A continuació veurem alguns exemples.

- Connect State: N3 inicia una connexió TCP amb port destí 179. N6 respon, i N3 confirma l'establiment de la sessió i envia un missatge Open (**Fig. 4.17**).

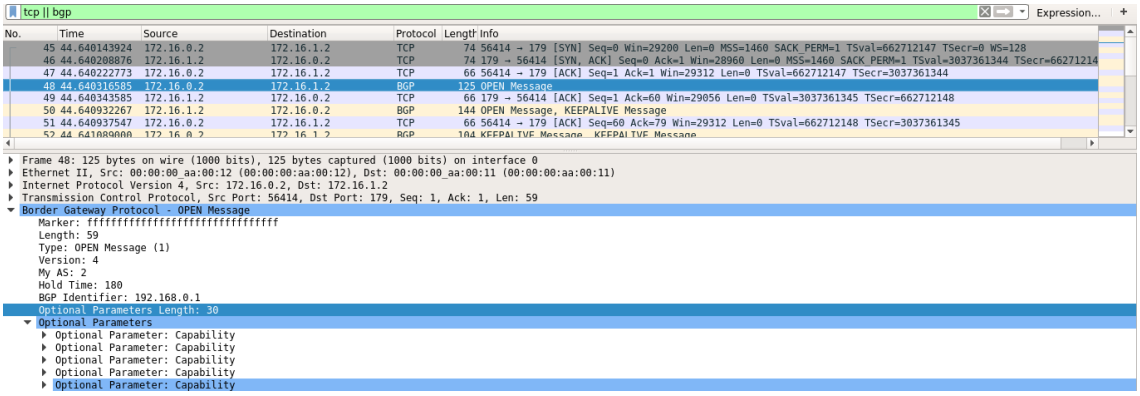

Fig. 4.17 Formació de veïnatge IBGP: missatge Open de N3 a N6

- Missatge Update: enviat de N3 a N6 per intercanviar informació d'encaminament, anuncia el prefix 10.0.0.0/16, i conté diversos atributs BGP (**Fig. 4.18**).

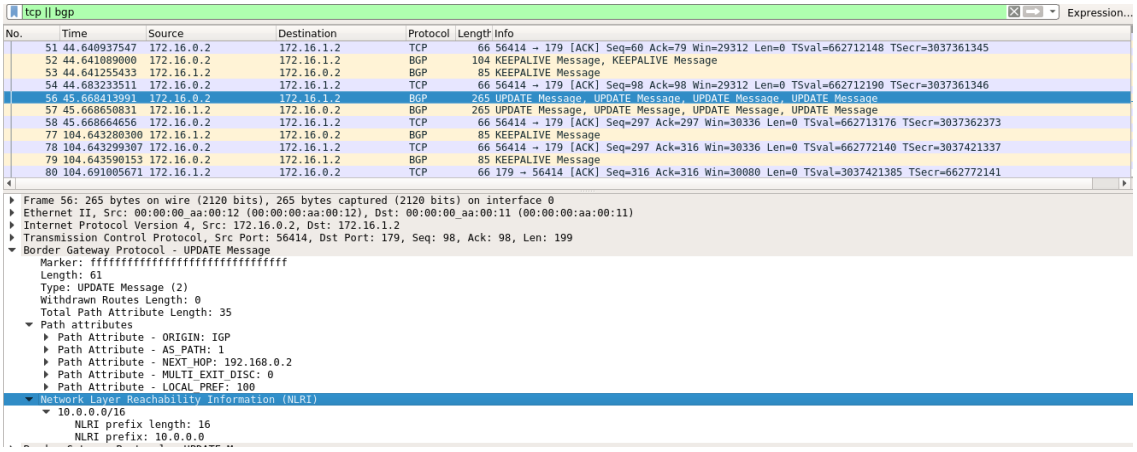

Fig. 4.18 Formació de veïnatge IBGP: missatge Update de N3 a N6

En aquest cas, CORE no incorpora widgets per veure l'estat de la formació de veïns BGP, així que personalitzarem un widget per tenir aquesta informació. Executarem la comanda apropiada per visualitzar la informació desitjada a la consola de Quagga (**Fig. 4.19**):

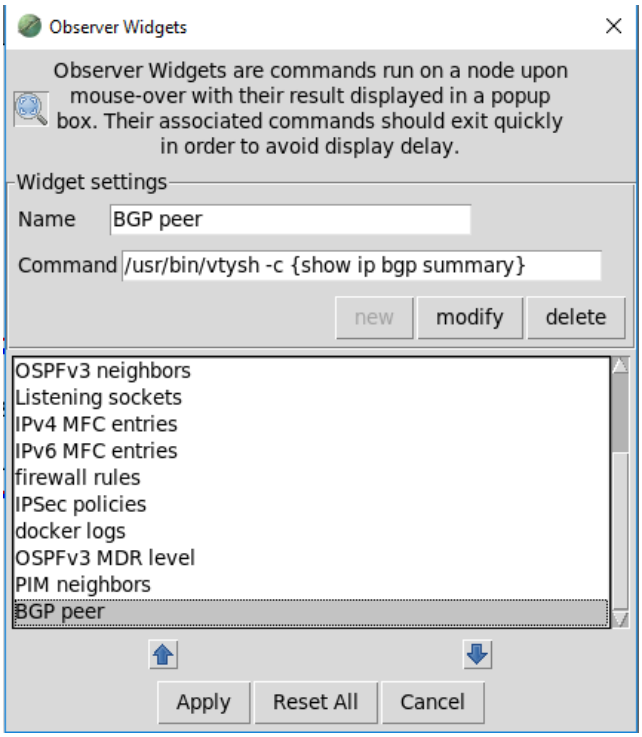

Fig. 4.19 Creació de widget de veïnatge BGP

Amb el widget creat podem veure informació dels veïns BGP (**Fig. 4.20**):

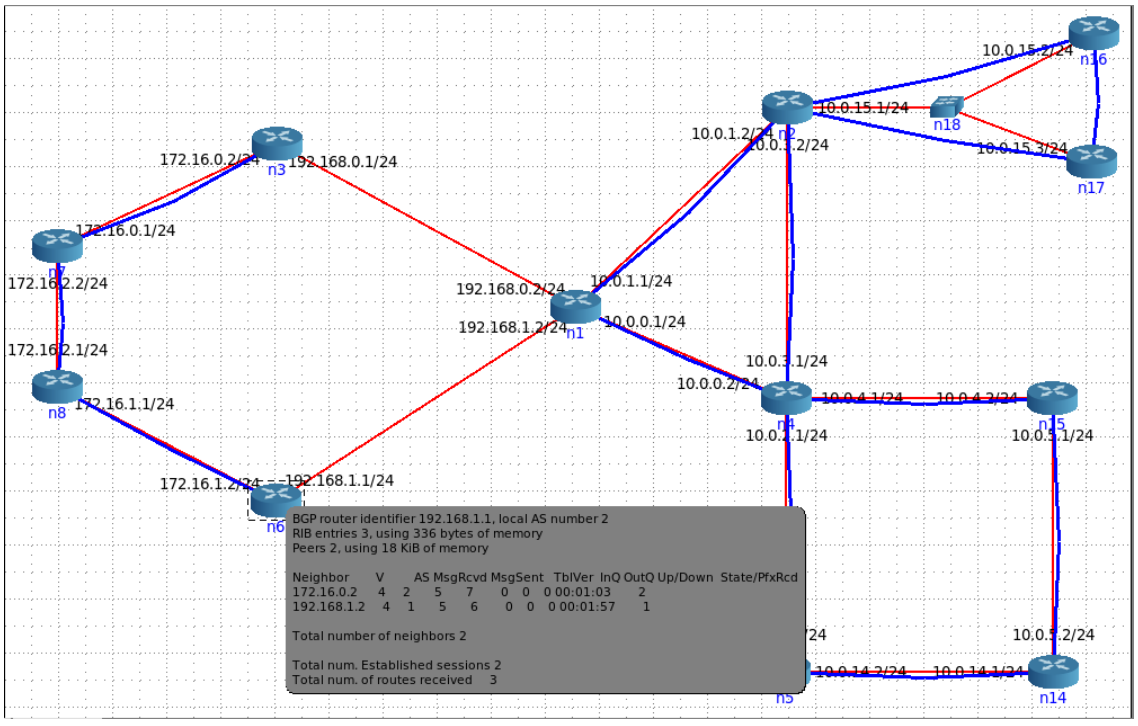

Fig. 4.20 Escenari amb el widget de veïnatge BGP creat

De nou podem accedir a la consola Quagga dels nodes per veure el correcte funcionament de l'escenari (**Fig. 4.21**):

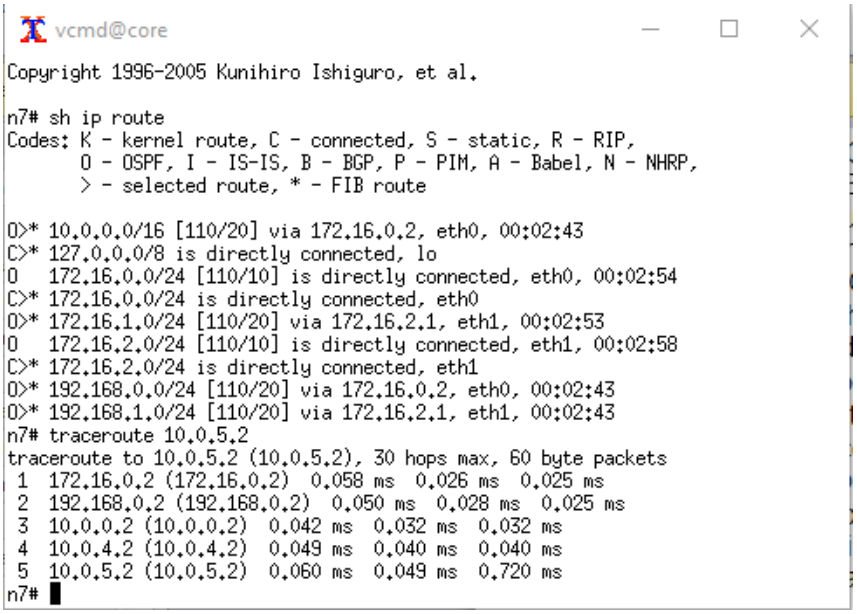

Fig. 4.21 Connectivitat des de N7 a N14, per N3 i N1

4.2. Desplegaments amb plantilles IMN: creació i configuració centralitzada de topologies

Fins ara hem vist el desplegament i la configuració dels elements emulats de manera individual; per tal d'aixecar els nodes (routers en l'escenari anterior) i carregar la configuració dels diferents serveis ("usr/local/etc/quagga/Quagga.conf", en l'escenari anterior) hem d'accedir al node en concret. Això comporta certes limitacions ja que, per escenaris amb diversos nodes, pot ser tediós un mètode distribuït de creació i configuració d'elements.

CORE utilitza dos plantilles; IMN i XML. En aquest apartat editarem l'arxiu IMN obtingut del escenari anterior per mostrar la possibilitat de desplegar elements i configurar diferents serveis a través d'un template. Configurarem diferents enllaços amb ample de banda limitat i pèrdues, i editarem el fitxer "/usr/local/etc/quagga/Quagga.conf" del N6 per incloure l'atribut BGP local-preference a 200. Posteriorment, obrirem el template amb CORE per comprovar el correcte funcionament.

La primera part de l'arxiu defineix els nodes. Inicialment, es defineix el tipus de node, les interfícies i els veïns connectats a cadascuna d'elles. A continuació, s'especifica la localització del node al mapa de la GUI, basada en Tcl/Tk. Aquest tret característic és probablement una contrapartida per crear i configurar grans topologies amb templates, ja que suposa una complexitat addicional per planificar l'escenari sense aportar valor a l'experiment. En cas de

ser irrellevant la il·lustració del escenari a través de la GUI, és possible establir tots els nodes en les mateixes coordenades.

Un cop definida la configuració bàsica del node, s'especifiquen els serveis que utilitzarem i la corresponent configuració. En el nostre cas, editarem l'arxiu "/usr/local/etc/quagga/Quagga.conf" del node 6 per afegir un local-preference de 200 a les rutes del prefix 10.0.0.0/16 (**Fig. 4.22**), rebut del node 1 per la interfície que els connecta. Així prioritzarem el camí a través de N6 per arribar a l'AS 1. També modificarem els enllaços N7-N3 i N3-N1 per limitar l'ample de banda a 67kbps i introduir un retard de 80ms (**Fig. 4.23**).

routingospfbgp4.imn

```
511     }
512     canvas c1
513     iconcoords {241.0 441.0}
514     labelcoords {241.0 473.0}
515     interface-peer {eth0 n8}
516     interface-peer {eth1 n1}
517     services {BGP OSPFv2 zebra IPForward}
518     custom-config {
519     custom-config-id service:zebra
520     custom-command zebra
521     config {
522     files=('/usr/local/etc/quagga/Quagga.conf', 'quaggaboot.sh', '/usr/local/etc/quagga/vtysh.conf', )
523     }
524     }
525     custom-config {
526     custom-config-id service:zebra:/usr/local/etc/quagga/Quagga.conf
527     custom-command /usr/local/etc/quagga/Quagga.conf
528     config {
529     interface eth0
530       ip address 172.16.1.2/24
531     !
532     interface eth1
533       ip address 192.168.1.1/24
534     !
535     !
536     ! BGP configuration
537     !
538     ! You should configure the AS number below,
539     ! along with this router's peers.
540     !
541     router bgp 2
542       bgp router-id 192.168.1.1
543       neighbor 192.168.1.2 remote-as 1
544       neighbor 172.16.0.2 remote-as 2
545       network 172.16.0.0/16
546       neighbor 192.168.1.2 route-map lp in
547     !
548     ip prefix-list myprefix permit 10.0.0.0/16
549     !
550     route-map lp permit 10
551       match ip adress prefix-list myprefix
552       set local-preference 200
553     !
```

Fig. 4.22 Configuració de N6 amb plantilles IMN

```
710
711 link l7 {
712     nodes {n7 n3}
713     delay 80000
714     bandwidth 64000
715 }
716
717 link l8 {
718     nodes {n8 n6}
719     bandwidth 0
720 }
721
722 link l9 {
723     nodes {n8 n7}
724     bandwidth 0
725 }
726
727 link l10 {
728     nodes {n3 n1}
729     delay 80000
730     bandwidth 64000
731 }
```

Fig. 4.23 Configuració del enllaços N7-N3 i N3-N1 amb plantilles IMN

A través del CLI, CORE permet desplegar plantilles IMN amb "*core-gui –batch*" (**Fig. 4.24**). Després podem accedir al node 7 amb vcmd, i executar un traceroute per veure quin camí segueix el tràfic cap a 10.0.5.2. Podem comprovar el correcte funcionament dels canvis introduïts; els paquets viatgen a través dels nodes 8 i 6.

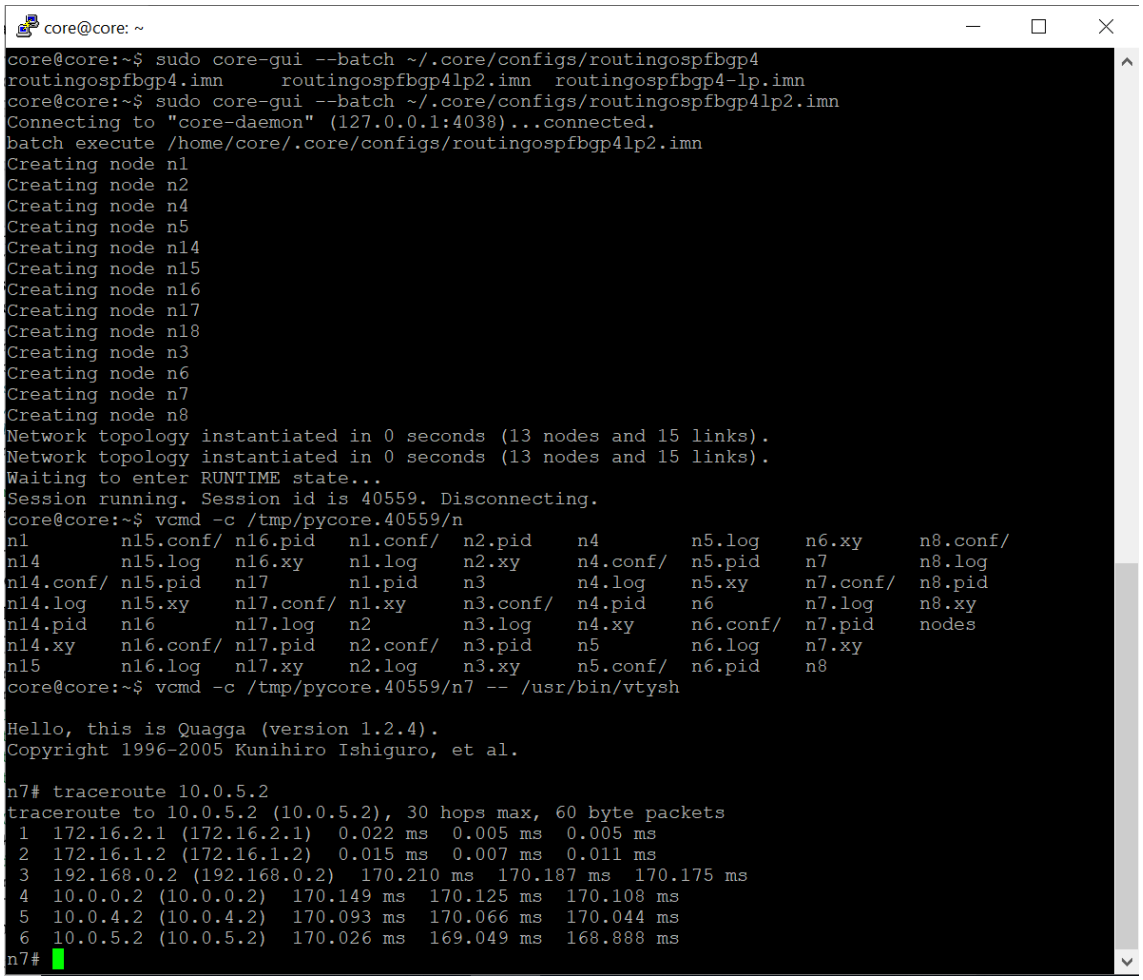

Fig. 4.24 Desplegaments amb plantilles IMN: efectes del local-preference

La documentació no recull el detall de les eines que CORE utilitza per manipular els enllaços. Fent una mica de recerca, podem veure com CORE fa ús de les eines TC (Traffic Control) i NetEm del nucli de Linux. El projecte de Linux NetEm és una millora del clàssic Linux Traffic Control. NetEm afegeix propietats de xarxa tals com latència, jitter, pèrdua o duplicació de paquets.

TC permet modelar, programar, classificar i prioritzar tràfic, habitualment, a la sortida de les interfícies:

- Modelatge: permet controlar la tassa de transmissió, controlant l'ample de banda disponible.
- Programació: a través de reordenar els paquets a transmetre en sortida, es possible prioritzar cert tràfic.
- Aplicació de polítiques: permet tractar i filtrar tràfic en entrada.
- Descartar tràfic: es possible descartar paquets en entrada i en sortida quan s'excedeix cert ample de banda.

El component bàsic de Linux Traffic Control és la disciplina de cues (qdisc), que permet establir un cert tractament del tràfic. Els qdiscs es poden concatenar a través de cadenes, o bé, en algunes modalitats, es poden relacionar a través

de classes formant relacions jeràrquiques entre ells. Per tal de classificar el tràfic en una classe, s'utilitzen filtres. Aquesta part surt del àmbit del projecte.

La implementació més senzilla d'un qdisc podria ser FIFO (first in first out, els primers paquets en arribar són els primers en sortir). La modalitat utilitzada per CORE és Token Bucket Filter (TBF), que configura el trànsit per ajustar-se a una velocitat de sortida desitjada. TBF està basat en la idea de tokens, que són utilitzats per cada paquet a transmetre. Cada paquet necessita de més d'un token, i en cas de no haver-hi disponibles, es posa en cua. Altres paràmetres utilitzats a CORE de TBF són:

- Burst: nombre màxim de bytes per els quals pot haver-hi tokens disponibles de manera instantània.
- Latency: temps màxim que un paquet pot estar en cua.

Utilitzant una cadena de qdisc, CORE estableix l'ample de banda en cada interfície del enllaç configurat a 64Kbit, i un retard de 80ms (aplicable només en sortida). CORE aplica totes les polítiques a l'extrem del veth que pertany al bridge localitzat al host d'emulació (**Fig. 4.25**). Al fer ping de N7 a N3, el RTT (round-trip time) és de 160ms aproximadament (**Fig. 4.26**).

```
core@core:~$ tc qdisc show dev
b.17852.85   b.38642.85   enp0s8       veth1.2.85   veth4.0.85   veth6.1.85   vethf.0.85
b.18348.85   b.48164.85   enp0s9       veth1.3.85   veth4.1.85   veth7.0.85   vethf.1.85
b.18.85      b.53021.85   lo           veth2.0.85   veth4.2.85   veth7.1.85
b.2218.85    b.64654.85   veth10.0.85  veth2.1.85   veth4.3.85   veth8.0.85
b.25096.85   b.65007.85   veth1.0.85   veth2.2.85   veth5.0.85   veth8.1.85
b.25296.85   b.9457.85    veth11.0.85  veth3.0.85   veth5.1.85   vethe.0.85
b.28950.85   enp0s3       veth1.1.85   veth3.1.85   veth6.0.85   vethe.1.85
core@core:~$ tc qdisc show dev veth7.0.85
qdisc tbf 1: root refcnt 2 rate 64Kbit burst 3000b lat 7.8s
qdisc netem 10: parent 1:1 limit 1000 delay 80.0ms
core@core:~$ tc qdisc show dev veth7.1.85
qdisc noqueue 0: root refcnt 2
core@core:~$ tc qdisc show dev veth3.0.85
qdisc tbf 1: root refcnt 2 rate 64Kbit burst 3000b lat 7.8s
qdisc netem 10: parent 1:1 limit 1000 delay 80.0ms
core@core:~$ tc qdisc show dev veth3.1.85
qdisc tbf 1: root refcnt 2 rate 64Kbit burst 3000b lat 7.8s
qdisc netem 10: parent 1:1 limit 1000 delay 80.0ms
core@core:~$ tc qdisc show dev veth1.
veth1.0.85  veth1.1.85  veth1.2.85  veth1.3.85
core@core:~$ tc qdisc show dev veth1.2.85
qdisc tbf 1: root refcnt 2 rate 64Kbit burst 3000b lat 7.8s
qdisc netem 10: parent 1:1 limit 1000 delay 80.0ms
core@core:~$
```

Fig. 4.25 Implementació del retard i l'ample de banda amb qdisc

```
core@core:~$ sudo vcmd -c /tmp/pycore.46757/n7 -- ping 172.16.0.2
PING 172.16.0.2 (172.16.0.2) 56(84) bytes of data.
64 bytes from 172.16.0.2: icmp_seq=1 ttl=64 time=160 ms
64 bytes from 172.16.0.2: icmp_seq=2 ttl=64 time=161 ms
64 bytes from 172.16.0.2: icmp_seq=3 ttl=64 time=161 ms
64 bytes from 172.16.0.2: icmp_seq=4 ttl=64 time=161 ms
64 bytes from 172.16.0.2: icmp_seq=5 ttl=64 time=161 ms
64 bytes from 172.16.0.2: icmp_seq=6 ttl=64 time=160 ms
64 bytes from 172.16.0.2: icmp_seq=7 ttl=64 time=161 ms
64 bytes from 172.16.0.2: icmp_seq=8 ttl=64 time=160 ms
^C
--- 172.16.0.2 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7001ms
rtt min/avg/max/mdev = 160.330/161.141/161.920/0.476 ms
core@core:~$
```

Fig. 4.26 Connectivitat entre N7 i N3: mesura del RTT

Per tal de mesurar l'ample de banda disponible, utilitzarem l'eina IPerf. CORE permet disposar de qualsevol servei als nodes si està instal·lat al host d'emulació. Per tal d'executar l'eina IPerf als nodes, utilitzarem la comanda vcmd. En les següents figures podem observar com l'ample de banda està limitat a 64Kbit/s. De nou, les restriccions s'apliquen a la interfície del veth que es troba localitzada al bridge que connecta els nodes. En canvi, l'eina IPerf està executada dintre del node, la qual cosa comporta que la restricció d'ample de banda s'aprecií al servidor (N3 en aquest cas).

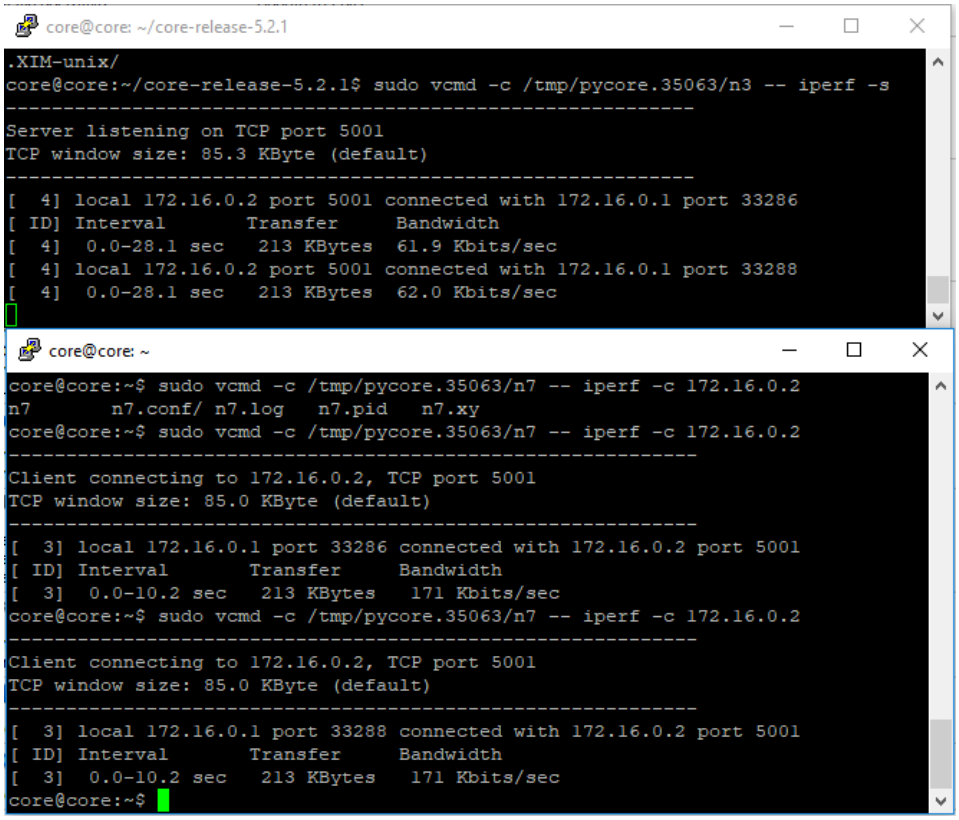

Fig. 4.27 Connectivitat entre N7 i N3: mesura del ample de banda

4.3. Escalabilitat: creació de topologies a través de CLI

Fins ara hem vist com emular escenaris amb un nombre de nodes relativament reduït. Aquests escenaris són molt útils per l'àmbit docent per exemple, de cara a entendre el funcionament de certs protocols o veure el comportament d'una topologia de xarxa sota determinades condicions d'enllaç. En altres ocasions, l'emulació de xarxes és utilitzada per crear grans topologies basades en nodes lleugers, per tal de disposar simplement d'una infraestructura amb connectivitat, i sobre la qual es treballarà per provar una aplicació determinada, per exemple.

Tal i com hem comentat anteriorment, CORE disposa d'una sèrie de comandes per jugar amb escenaris basats en LXC des de CLI:

- vnoded: dimoni llançat per crear contenidors i escoltar comandes a través del socket que exerceix com a canal de control (/tmp/pycore."sessió"/"node" o /tmp/"node".ctl). És executat sota el PID 1 al contenidor, i trobarem un procés vnoded al host d'emulació per cada node creat.
- vcmd: comanda per connectar al dimoni vnoded, que corre sobre el contenidor per tal de poder executar comandes.

A través d'aquestes comandes i de les diferents eines que sabem que CORE utilitza per crear els escenaris, podem explotar les capacitats de CORE per crear grans topologies. En el capítol anterior, per exemple, hem vist com CORE utilitza Linux Traffic Control per modificar els enllaços. També sabem que CORE utilitza Linux Bridge per connectar els nodes. Aquest apartat també pot servir per determinar la compatibilitat amb OVS com a opció per connectar els nodes, la qual cosa no apareix a la documentació. Combinant les diferents eines esmentades, ens ajudarem d'uns scripts creats per desplegar dos topologies diferents. En ambdós casos, la creació dels nodes i els enllaços segueix les següents pautes:

1. Creació dels nodes amb vnoded. S'assigna una canal de control, un arxiu de logs i un PID. Desafortunadament, no ha estat possible recollir logs dels nodes mitjançant aquest fitxer. CORE disposa d'un arxiu (/var/log/core-daemon.log) de logs al host d'emulació, al igual que qualsevol aplicació Linux, des d'on és possible recollir qualsevol esdeveniment relacionat amb les diferents emulacions: posada en marxa del dimoni core, desplegament de topologies o interacció amb la GUI, entre altres.

   Per altra banda, quan es desplega una topologia per la GUI, el canal de control i els logs d'un node es troben sota /tmp/pycore."sessió"/"node", i en aquest cas si ha estat possible visualitzar els logs d'un node, però només s'inclouen esdeveniments relacionats amb el procés vnoded (PID 1 al espai de noms del node). Això inclou qualsevol interacció entre el node i el host d'emulació, però no trobarem aspectes exclusius del node, com l'execució d'una aplicació en aquest.

2. Creació d'un veth entre una interfície del bridge de connexió i el node.

3. Assignació de la interfície del veth pertanyent al extrem del node al espai de noms de xarxa del contenidor.

4. Configuració del node a través de vcmd. S'assigna una IP a la interfície que connecta al bridge de connexió (172.16.0.0./16), i s'estableixen dos qdisc per controlar l'ample de banda i el retard. Es fixa el valor màxim de ràfega (burst) en 32 kbit i s'admeten paquets amb una latència màxima d'un segon.

CORE ja inclou un script per matar tots els processos vnoded des de CLI.

4.3.1. Topologia simple

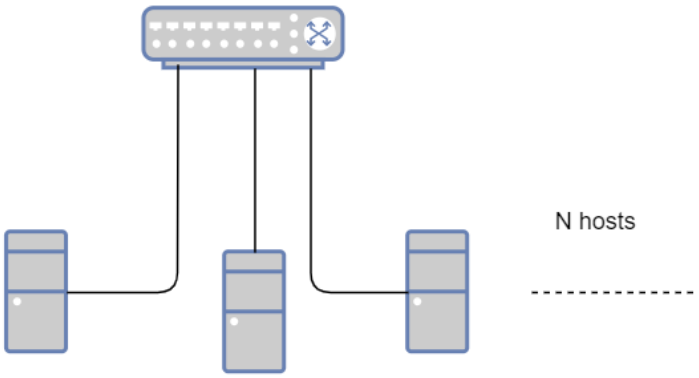

Fig. 4.28 Topologia simple: un switch OVS i N hosts

En aquest primer cas hem utilitzat OVS com a solució de switching per connectar nodes. El funcionament ha estat satisfactori, i seria una funcionalitat interesant per la GUI en les següents actualitzacions de CORE. Recordem que OVS incorpora suport per una amplia llista de protocols, a diferencia de Linux Bridge.

linear-topob.bash | simple-topob.bash

```
#/bin/bash
ovs-vsctl add-br ovsbr0
ifconfig ovsbr0 up
y=0
z=2
for ((i=1;i<=$1 && y<256;i++))

    do
    vnoded -c /tmp/n$i.ctl -l /tmp/n$i.log -p /tmp/n$i.pid
    ip link add name n$i.br type veth peer name n$i.node
    ip link set n$i.node netns `cat /tmp/n$i.pid`
    vcmd -c /tmp/n$i.ctl -- ip link set lo up
    vcmd -c /tmp/n$i.ctl -- ip link set n$i.node name eth0 up
    vcmd -c /tmp/n$i.ctl -- ip addr add 172.16.$y.$z/16 dev eth0
    z=$((z+1))
    if (($z>254))
    then
            y=$((y+1))
            z=2
    fi
    vcmd -c /tmp/n$i.ctl -- tc qdisc add dev eth0 root handle 1: tbf rate $2bit burst 32kbit latency 1000ms
    vcmd -c /tmp/n$i.ctl -- tc qdisc add dev eth0 parent 1:1 handle 10: netem delay $3ms
    ovs-vsctl add-port ovsbr0 n$i.br
    ip link set n$i.br up
    done

ovs-vsctl show

```

Fig. 4.29 Script per la creació de la topologia simple

Inicialment tenim el host d'emulació utilitzant 211MB de memòria, així com una carrega de CPU relativament baixa.

```
core@core:~/core-release-5.2.1$ free -m
              total        used        free      shared  buff/cache   available
Mem:           3000         211        2257           1         532        2626
Swap:          2047           0        2047
core@core:~/core-release-5.2.1$ top
top - 19:25:16 up 3 min,  1 user,  load average: 0.36, 0.25, 0.11
Tasks: 179 total,   1 running,  68 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.1 sy,  0.0 ni, 99.9 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  3072924 total,  2310548 free,   216788 used,   545588 buff/cache
KiB Swap:  2097148 total,  2097148 free,        0 used.  2689112 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
12968 core      20   0   42784   3904   3304 R   0.5  0.1   0:02.30 top
```

Fig. 4.30 Recursos utilitzats inicialment

A la documentació de CORE es parla de l'emulador com a escalable i eficient, i es citen una sèrie d'objectius i no objectius. Com a objectiu, s'estableix que CORE ha de ser capaç d'executar desenes de nodes en un ordinador de pocs recursos. Com a no objectius, es fa referencia a no establir un nombre màxim de nodes, i que el límit ha de ser realista d'acord als recursos disponibles. En el nostre PC, desplegarem 100 hosts connectats al switch ovsbr0, amb un ample de banda de 50Kbps i 80 ms de retard.

```
46a1fec5-fe3a-4bcd-90d2-40995cb4bbd2
    Bridge "ovsbr0"
        Port "n1.br"
            Interface "n1.br"
        Port "n14.br"
            Interface "n14.br"
        Port "n49.br"
            Interface "n49.br"
        Port "n44.br"
            Interface "n44.br"
        Port "n27.br"
            Interface "n27.br"
        Port "n84.br"
            Interface "n84.br"
        Port "n8.br"
            Interface "n8.br"
        Port "n17.br"
            Interface "n17.br"
```

Fig. 4.31 Creació de la topologia simple amb 100 nodes

Amb "*ip -o link | grep ".br*"" podem veure els enllaços entre les interfícies del bridge OVS i la interfície assignada a l'espai de noms de xarxa propi del node (**Fig. 4.32**).

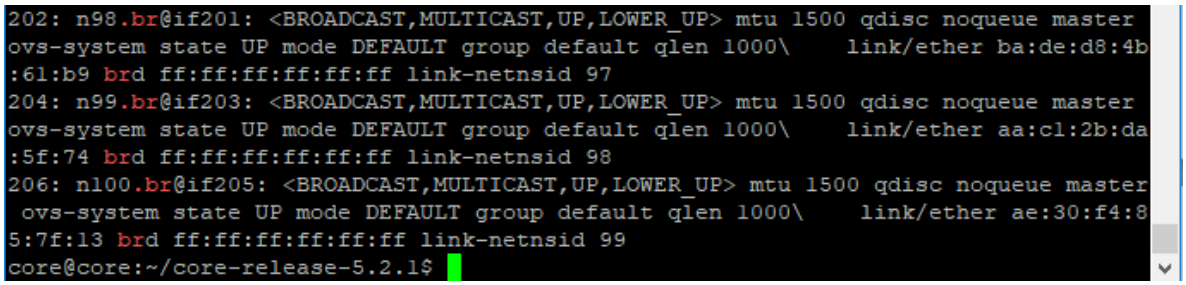

Fig. 4.32 Creació de la topologia simple: "*ip -o link | grep ".br"*"

En la següent captura podem observar el funcionament de la connectivitat entre hosts, amb els paràmetres definits d'enllaç (**Fig. 4.33**). Es realitza un ping entre N1 i N2 per comprovar el RTT teòric de 160ms, i s'utilitza l'eina IPerf per comprovar l'ample de banda entre els dos nodes.

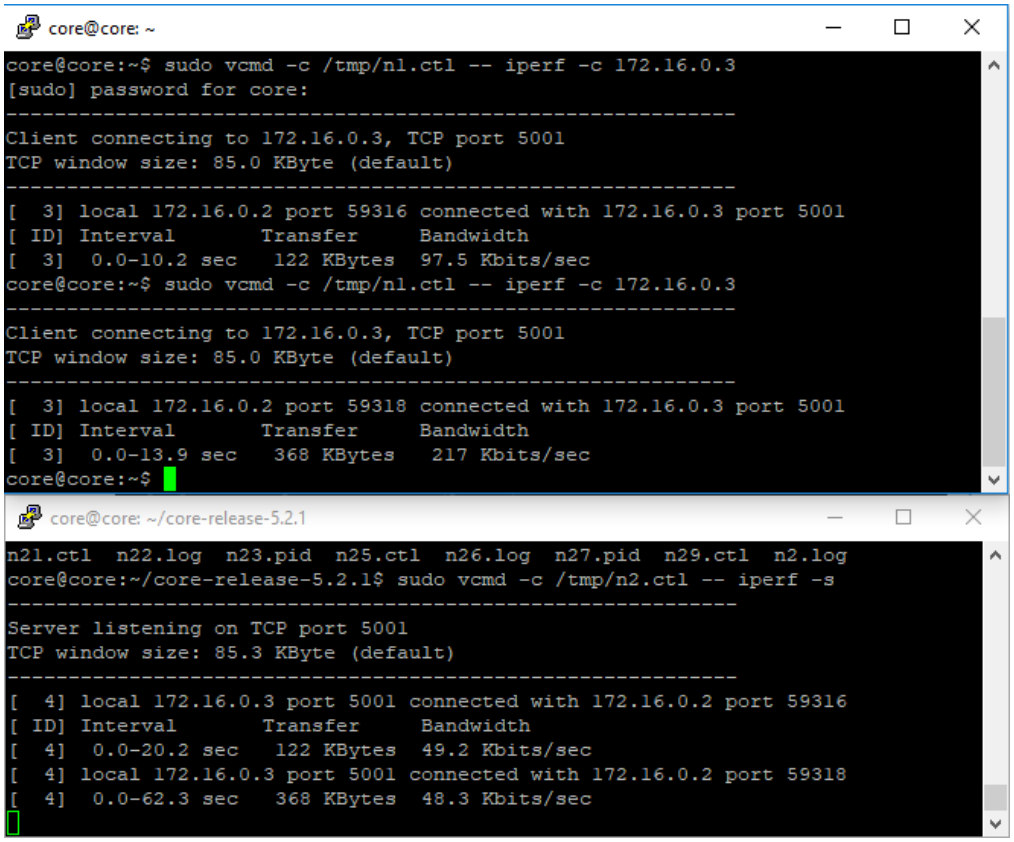

Fig. 4.33 Creació de la topologia simple: connectivitat entre N1 i N2

A partir d'aquets punt, hem intentat crear diferents escenaris amb 500 (**Fig. 4.35**) i 1000 (**Fig. 4.37**) nodes en el nostre equip per tal de determinar, amb els recursos disponibles, si s'obtenen desplegaments satisfactoris.

```
core@core:~/core-release-5.2.1$ free -m
              total        used        free      shared  buff/cache   available
Mem:           3000         269        2179           2         552        2564
Swap:          2047           0        2047
core@core:~/core-release-5.2.1$ top
top - 19:48:55 up 26 min,  1 user,  load average: 0.00, 0.02, 0.06
Tasks: 219 total,   1 running, 168 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.2 sy,  0.0 ni, 99.6 id,  0.1 wa,  0.0 hi,  0.1 si,  0.0 st
KiB Mem :  3072924 total,  2233256 free,   273680 used,   565988 buff/cache
KiB Swap:  2097148 total,  2097148 free,        0 used.  2628532 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
26205 root      20   0   10432   1456   1328 S   0.0  0.0   0:00.00 vnoded
26266 root      20   0   10432   1496   1372 S   0.0  0.0   0:00.00 vnoded
26328 root      20   0   10432   1460   1336 S   0.0  0.0   0:00.00 vnoded
26389 root      20   0   10432   1472   1352 S   0.0  0.0   0:00.00 vnoded
26469 core      20   0   42920   4088   3456 R   0.0  0.1   0:06.06 top
```

Fig. 4.34 Recursos utilitzats per la creació de 100 nodes

```
core@core:~/core-release-5.2.1$ free -m
              total        used        free      shared  buff/cache   available
Mem:           3000         530        1881           5         588        2287
Swap:          2047           0        2047
core@core:~/core-release-5.2.1$ top
top - 20:04:11 up 42 min,  1 user,  load average: 0.01, 0.17, 0.25
Tasks: 619 total,   1 running, 568 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.6 us,  1.6 sy,  0.0 ni, 97.8 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  3072924 total,  1923648 free,   544716 used,   604560 buff/cache
KiB Swap:  2097148 total,  2097148 free,        0 used.  2340648 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 2910 Debian-+  20   0   66380  14044   7092 S   7.9  0.5   1:23.78 snmpd
25971 core      20   0   43316   4652   3428 R   0.7  0.2   0:09.13 top
```

Fig. 4.35 Recursos utilitzats per la creació de 500 nodes

Al intentar crear més de 500 hosts, els procés del switch OVS utilitza molt CPU i queda saturat, alentint molt el desplegament (**Fig. 4.36**):

```
core@core:~$ free -m
              total        used        free      shared  buff/cache   available
Mem:           3000         725        1643           7         632        2081
Swap:          2047           0        2047
core@core:~$ top
top - 20:11:26 up 49 min,  2 users,  load average: 4.14, 2.08, 0.97
Tasks: 912 total,   2 running, 853 sleeping,   0 stopped,   0 zombie
%Cpu(s): 33.7 us, 22.0 sy,  0.0 ni, 43.4 id,  0.0 wa,  0.0 hi,  1.0 si,  0.0 st
KiB Mem :  3072924 total,  1673400 free,   749864 used,   649660 buff/cache
KiB Swap:  2097148 total,  2097148 free,        0 used.  2123040 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 1585 root      10 -10  407020  73320   8912 R 109.9  2.4   5:43.60 ovs-vswitc+
  832 systemd+  20   0   71548   6680   5400 S  36.5  0.2   2:10.56 systemd-re+
 2910 Debian-+  20   0   68056  15644   7092 S  21.4  0.5   1:56.32 snmpd
 1256 root      10 -10   31412  15112   4008 S   7.9  0.5   0:34.30 ovsdb-serv+
 1006 root      20   0  170400  18624   9476 S   1.3  0.6   0:07.87 networkd-d+
14313 core      20   0   43612   4816   3316 R   1.3  0.2   0:00.09 top
```

Fig. 4.36 Recursos utilitzats per la creació de més de 500 nodes

```
core@core:~/core-release-5.2.1$ free -m
              total        used        free      shared  buff/cache   available
Mem:           3000         860        1471           9         669        1937
Swap:          2047           0        2047
core@core:~/core-release-5.2.1$ top
top - 20:25:35 up  1:03,  2 users,  load average: 0.94, 2.36, 1.91
Tasks: 1127 total,   1 running, 1070 sleeping,   0 stopped,   0 zombie
%Cpu(s):  1.2 us,  0.5 sy,  0.0 ni, 98.3 id,  0.0 wa,  0.0 hi,  0.1 si,  0.0 st
KiB Mem :  3072924 total,  1505304 free,   882312 used,   685308 buff/cache
KiB Swap:  2097148 total,  2097148 free,        0 used.  1982528 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 1585 root      10 -10  411112  80124   8912 S   3.3  2.6  15:14.19 ovs-vswitchd
 1256 root      10 -10   34140  18124   4008 S   2.3  0.6   1:24.87 ovsdb-server
 4769 core      20   0   44012   5232   3440 R   1.3  0.2   0:00.33 top
```

Fig. 4.37 Recursos utilitzats per la creació de 1000 nodes

A mesura que augmenta el nombre de hosts, augmenta de forma no lineal la memòria utilitzada del host. A partir de 1000 hosts, el temps de desplegament al nostre host és excessiu degut a la carrega de CPU.

4.3.2. Topologia lineal

En aquesta topologia es desplega un nombre determinat de switchs (Linux Bridge) en cascada, interconnectats entres si, i amb un host per switch. També inclou definició d'ample de banda i retard. Es pot consultar l'script a ANNEX 1.

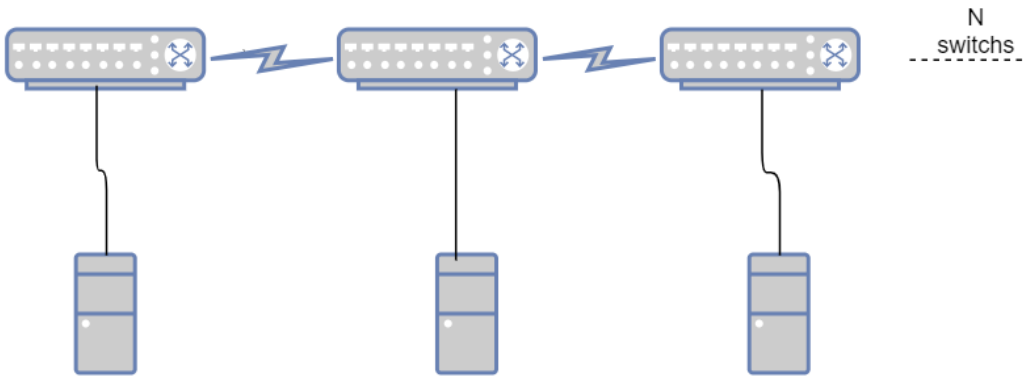

Fig. 4.38 Topologia lineal: N switchs i N hosts

```
bridge name     bridge id               STP enabled     interfaces
br1             8000.b2fb9f1f3bf6       no              br1.int.right
                                                        n1.br
br10            8000.66556d4d8da1       no              br10.int.left
                                                        br10.int.right
                                                        n10.br
br100           8000.829c16119989       no              br100.int.left
                                                        n100.br
br11            8000.2e68e258178d       no              br11.int.left
                                                        br11.int.right
                                                        n11.br
br12            8000.222aa3578379       no              br12.int.left
                                                        br12.int.right
                                                        n12.br
br13            8000.421f62b1e36f       no              br13.int.left
                                                        br13.int.right
                                                        n13.br
br14            8000.1e85385d654c       no              br14.int.left
                                                        br14.int.right
                                                        n14.br
br15            8000.362a975b0cec       no              br15.int.left
                                                        br15.int.right
                                                        n15.br
```

Fig. 4.39 Creació de la topologia lineal amb 100 nodes

```
core@core:~/core-release-5.2.1$ free -m
              total        used        free      shared  buff/cache   available
Mem:           3000         849        1320          16         831        1938
Swap:          2047           0        2047
core@core:~/core-release-5.2.1$ top
top - 22:15:34 up  2:53,  2 users,  load average: 3.69, 5.58, 3.42
Tasks: 631 total,   1 running, 570 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.1 us,  0.1 sy,  0.0 ni, 84.6 id,  0.0 wa,  0.0 hi, 15.2 si,  0.0 st
KiB Mem :  3072924 total,  1351568 free,   870320 used,   851036 buff/cache
KiB Swap:  2097148 total,  2097148 free,        0 used.  1984488 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
   22 root      20   0       0      0      0 S  17.7  0.0   3:15.29 ksoftirqd/2
   16 root      20   0       0      0      0 S  16.7  0.0   3:17.14 ksoftirqd/1
   28 root      20   0       0      0      0 S  14.8  0.0   3:12.12 ksoftirqd/3
    7 root      20   0       0      0      0 S  14.4  0.0   3:26.22 ksoftirqd/0
    8 root      20   0       0      0      0 I   1.3  0.0   0:34.00 rcu_sched
27741 core      20   0   43316   4728   3512 R   0.7  0.2   0:00.24 top
```

Fig. 4.40 Recursos utilitzats per la creació de 500 nodes

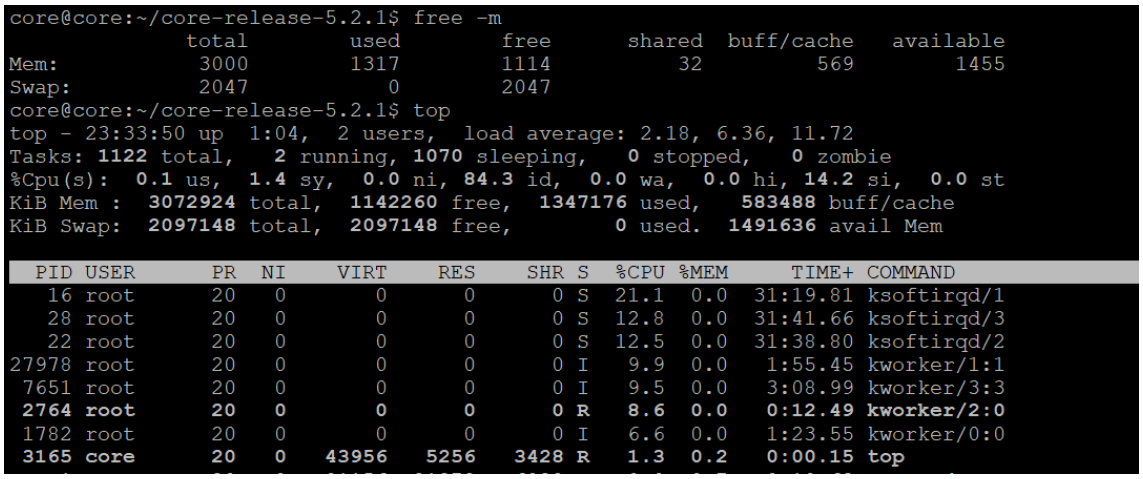

Fig. 4.41 Recursos utilitzats per la creació de 1000 nodes

En aquest escenari, el consum de memòria és lleugerament major al utilitzar múltiples bridges. Tot i utilitzar la seva pròpia eina per gestionar els nodes, el procés principal de CORE i els diferents vnoded es mantenen sempre estables i amb un consum baix de recursos en ambdós escenaris. En aquest experiment hem utilitzat una configuració bàsica als nodes per proporcionar connectivitat, i aquests valor poden augmentar depenent de la utilització dels nodes.

Al desplegar escenaris amb més de 500 nodes, diferents processos ksoftirqd consumeixen gran CPU (**Fig. 4.41**). Linux, així com múltiples SO, utilitza interrupcions de software i hardware. Aquestes senyals són enviades des de l'espai d'usuari al nucli del SO per informar d'un succés, resultant en un canvi de la seqüència d'instruccions executada per la CPU. Quan la CPU rep múltiples senyals d'interrupció seguides i no pot gestionar-les, el procés ksoftirqd (un per CPU) s'encarrega de posar-les en cua. Linux utilitza softirqs (un tipus d'interrupcions de software) per gestionar les estructures de paquets o per navegar entre la pila de xarxa al rebre'n.

4.4. Serveis SDN basats en OpenFlow: OVS i controlador RYU

En experiments anteriors, hem vist com implementar serveis clàssics de xarxa com OSPF o BGP, els quals estan suportats per una amplia gama d'emuladors. En els últims anys, les xarxes SDN (Software Defined Networking) són una realitat. A través de la separació del pla de control del pla de dades, s'obtenen múltiples beneficis tals com una visió completa de la xarxa i un control centralitzat. En l'última actualització de CORE, es van incloure dos serveis per suportar xarxes SDN basades en OpenFlow: OVS i controlador Ryu. A través d'aquests serveis, és possible aixecar switchs OpenFlow amb OVS en la nostra topologia, i connectar-los a un controlador Ryu per afegir fluxos a la taula de commutació. Al ser serveis nous, CORE no incorpora documentació amb consells o exemples per construir les topologies habituals, per la qual cosa la seva utilització pot ser confusa. En els següents escenaris tractarem d'il·lustrar

i documentar el funcionament d'aquests serveis a CORE, fent èmfasis en la integració amb els altres nodes de l'escenari i el host d'emulació.

El primer pas per tal de disposar dels serveis OVS i Ryu és la instal·lació d'aquests al host d'emulació. Amb l'actualització del host, podrem observar dos afegits a CORE: un node OVS i un servei OVS. El node OVS serveix per crear un switch que sols funciona amb aquest servei aixecat. A més, trobarem un servei OVS disponible per tots els nodes, el qual permet la integració del servei en qualsevol node.

4.4.1. Controlador en node local

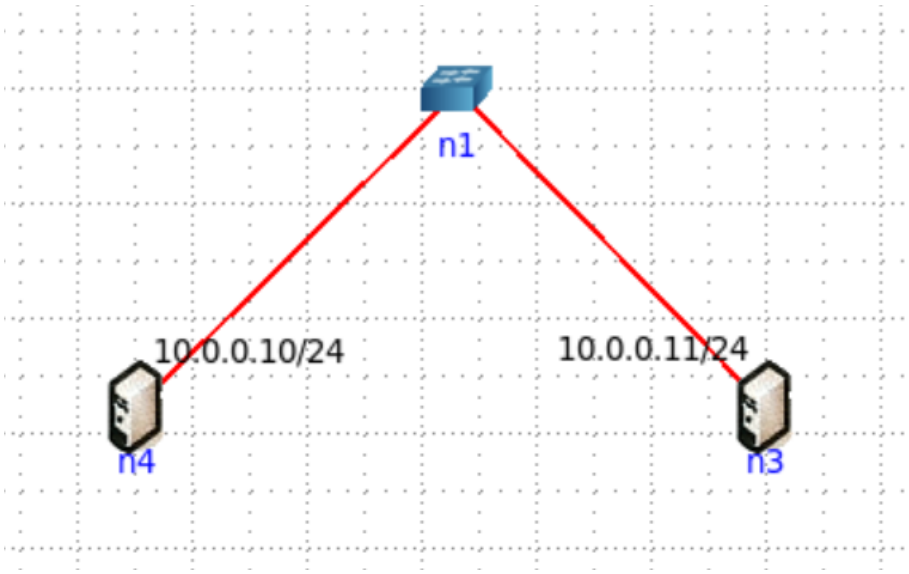

Fig. 4.42 Topologia SDN: controlador OpenFlow i OVS en local

La primera de les topologies que provarem és senzilla: un node amb 2 OVS switchs connectats per un veth, amb ambdós accedint en local (mateix node) al controlador Ryu per completar la taula de fluxos. Per defecte, CORE configura els serveis OVS per intentar connectar amb un controlador SDN local, i assigna totes les interfícies al switch OVS iniciat. La configuració del servei OVS es fa seguint les comandes habituals proporcionades per l'eina (**Fig. 4.43**).

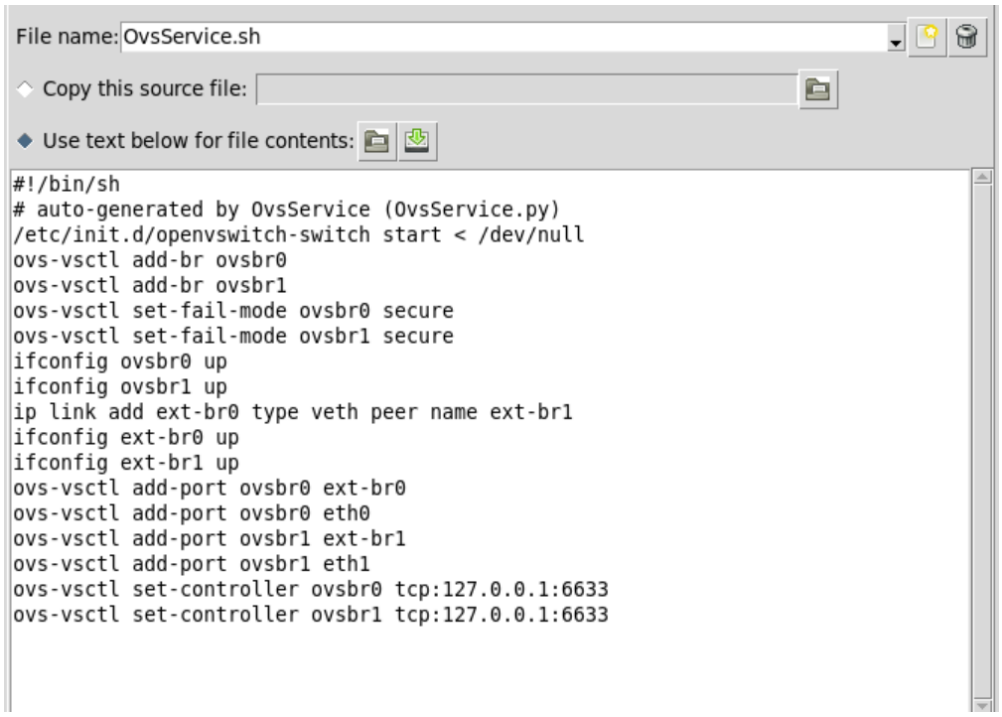

Fig. 4.43 Configuració del servei OVS

A continuació, hem carregat un mòdul senzill a Ryu per a que el switch OVS funcioni com a switch convencional (simple_switch_13.py) aprenent adreces MAC (**Fig. 4.44**).

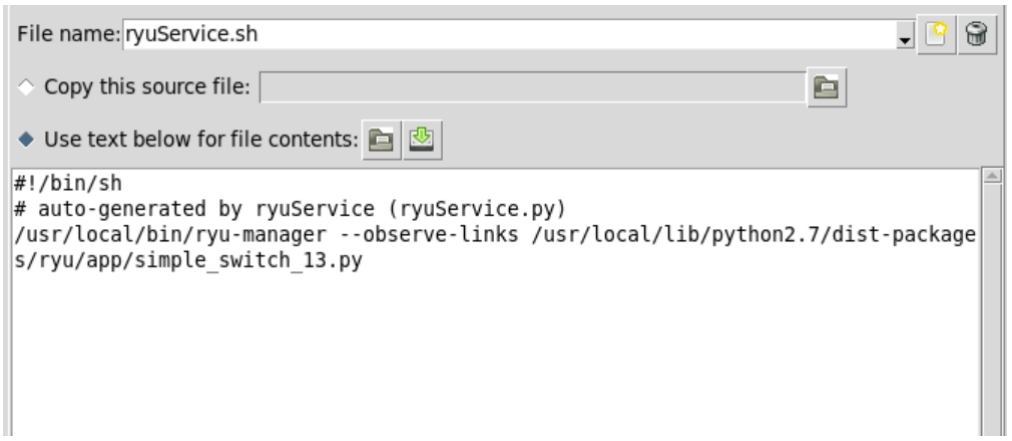

Fig. 4.44 Configuració del controlador Ryu

Inicialment trobem una sola regla a la taula de cada switch, on s'indica la connexió al controlador per defecte. Al iniciar un ping entre els hosts N4 i N3, els paquets arriben als switchs OVS i connecten amb el controlador OpenFlow, el qual instal·la dos regles per commutar els paquets entre les interfícies dels switchs creades prèviament (**Fig. 4.45**).

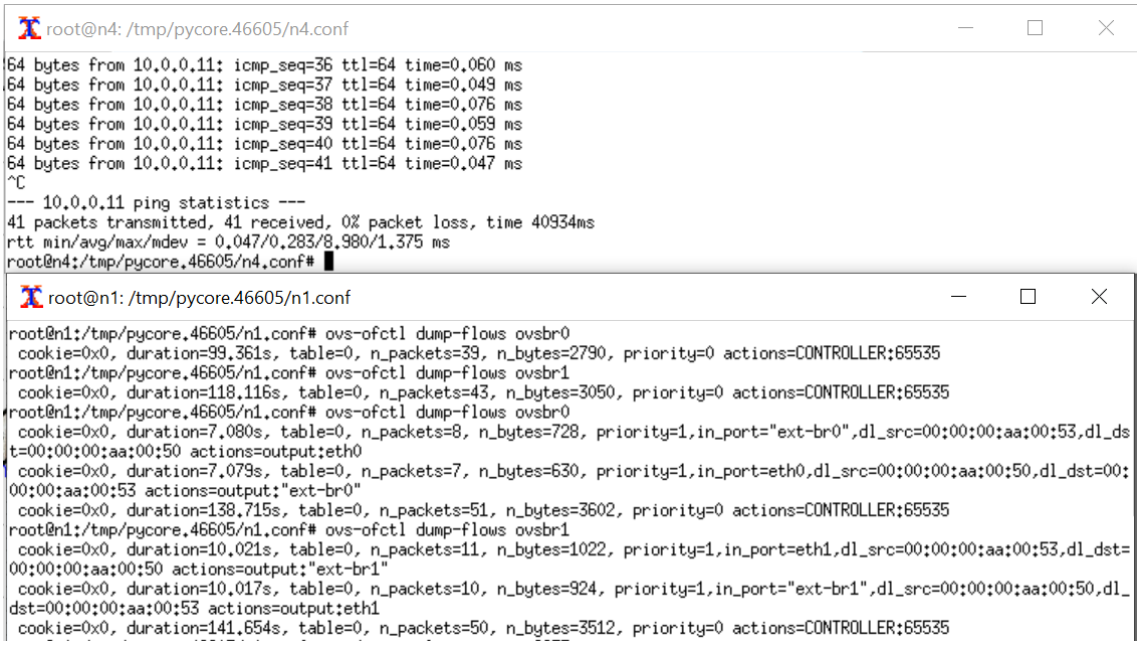

Fig. 4.45 Configuració del controlador Ryu

4.4.2. Controlador en node extern

En aquest escenari mostrarem com connectar els nodes OVS a un controlador extern, localitzat en un node també emulat a CORE. A CORE, el servei OVS es crea en un node, per la qual cosa queda aïllat en el seu propi espai de noms, i és necessari establir accés entre el node i el controlador extern OpenFlow. En cas de desplegar una topologia amb un nombre elevat de switchs OVS, i voler controlar-los mitjançant un controlador extern, serà necessari donar connectivitat cap al controlador des de cadascun dels nodes amb OVS. Això comporta una tasca addicional de configuració, necessitant múltiples adreces IPs i veths per realitzar la connexió. En aquest àmbit, CORE difereix d'altres emuladors SDN com Mininet, on el servei OVS es crea al root namespace **3.4.2**, i només és necessari establir accés des del host d'emulació al controlador. Per tant, de la mateixa manera que hem implementat la nostra topologia en l'apartat **4.3.1**, seria adient incloure el servei OVS com a opció per connectar nodes al root namespace, obtenint així una experiència més pràctica i escalable.

Abans d'entendre el funcionament, vam realitzar diversos experiments sense èxit, tractant de buscar formes de donar connectivitat entre els switchs i el controlador. A més, vam buscar ajuda a la comunitat de GitHub, de nou sense èxit, on diversos usuaris prèviament havien mostra't el seu interès en aquest tipus de topologies. Degut a la confusió causada en la comunitat de CORE en la seva implementació, el servei OVS ha estat eliminat de cara a futures versions de l'emulador.

Els objectius plantejats per tal d'obtenir l'escenari desitjat seran:

1. Construir una topologia habitual de dos switchs i 2 hosts (**Fig. 4.46**).

2. Configurar els switchs amb el servei OVS, incloent al servei les interfícies que connecten els switchs entre ells i les que donen accés als hosts.

3. Establir un enllaç des dels dos switchs a un tercer host, on estarà allotjat el controlador Ryu. Aquest tercer enllaç, en ambdós switchs, no estarà inclòs al servei OVS, i sols servirà per donar connectivitat amb el controlador OpenFlow al rebre els paquets procedents de N3 o N4 (**Fig. 4.48**).

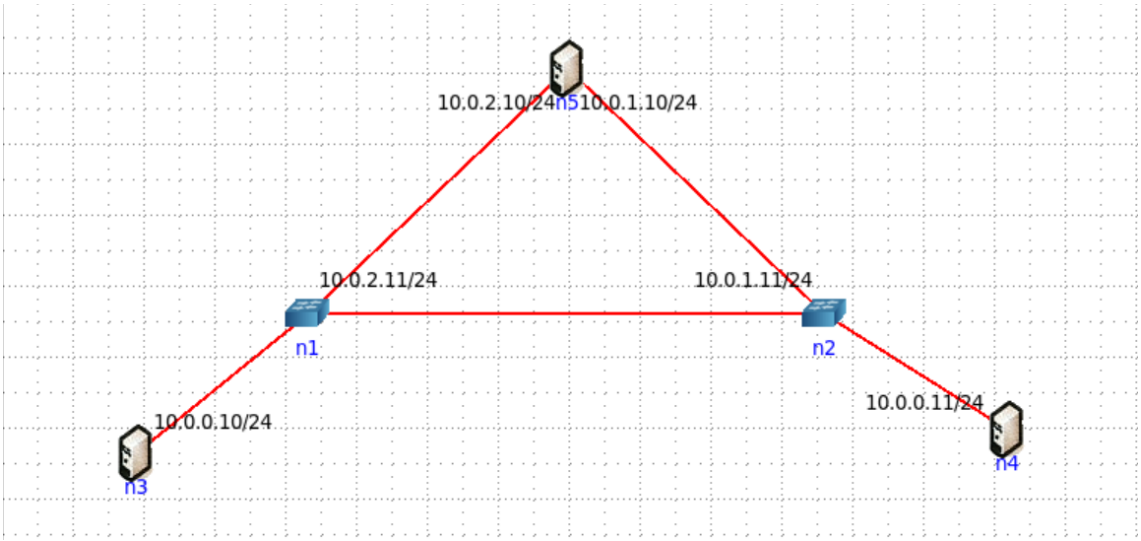

Fig. 4.46 Topologia SDN: controlador OpenFlow extern

A continuació es mostra la configuració del servei OVS del node N1 com a exemple (**Fig. 4.47**). N1 té tres interfícies; dos interfícies formen part del servei OVS. Aquestes dos interfícies, eth0 i eth2, connecten al switch N2 i al host N3, i la seva configuració és la mateixa que en l'escenari anterior. La interfície eth1 no forma part del servei OVS, i està configurada amb una IP (10.0.2.11) per poder accedir al controlador Ryu, present en el node N5.

File name: OvsService.sh

Copy this source file:

Use text below for file contents:

```
#!/bin/sh
# auto-generated by OvsService (OvsService.py)
/etc/init.d/openvswitch-switch start < /dev/null
ovs-vsctl add-br ovsbr0
ovs-vsctl set-fail-mode ovsbr0 secure
ifconfig ovsbr0 up
ovs-vsctl add-port ovsbr0 eth0
ovs-vsctl add-port ovsbr0 eth2
ovs-vsctl set-controller ovsbr0 tcp:10.0.2.10:6633
```

Fig. 4.47 Configuració del servei OVS

root@n3: /tmp/pycore.37529/n3.conf

```
root@n3:/tmp/pycore.37529/n3.conf# ping 10.0.0.11
PING 10.0.0.11 (10.0.0.11) 56(84) bytes of data.
64 bytes from 10.0.0.11: icmp_seq=1 ttl=64 time=36.1 ms
64 bytes from 10.0.0.11: icmp_seq=2 ttl=64 time=0.078 ms
64 bytes from 10.0.0.11: icmp_seq=3 ttl=64 time=0.075 ms
64 bytes from 10.0.0.11: icmp_seq=4 ttl=64 time=0.857 ms
^C
--- 10.0.0.11 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3071ms
rtt min/avg/max/mdev = 0.075/9.282/36.121/15.498 ms
root@n3:/tmp/pycore.37529/n3.conf#
```

root@n1: /tmp/pycore.37529/n1.conf

```
root@n1:/tmp/pycore.37529/n1.conf# ovs-ofctl dump-flows ovsbr0
 cookie=0x0, duration=33.007s, table=0, n_packets=22, n_bytes=1540, priority=0 actions=CONTROLLER:65535
root@n1:/tmp/pycore.37529/n1.conf# tcpdump -i eth1 tcp port 6633
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
^C20:40:24.195817 IP 10.0.2.11.60404 > 10.0.2.10.6633: Flags [P.], seq 772380246:772380358, ack 1724135958, w
in 229, options [nop,nop,TS val 2884953978 ecr 1701657068], length 112: OpenFlow
        version unknown (0x04), type 0x0a, length 112, xid 0x00000000
20:40:24.200900 IP 10.0.2.10.6633 > 10.0.2.11.60404: Flags [P.], seq 1:111, ack 112, win 235, options [nop,no
p,TS val 1701659118 ecr 2884953978], length 110: OpenFlow
        version unknown (0x04), type 0x0d, length 110, xid 0x0ba5a645
20:40:24.201456 IP 10.0.2.11.60404 > 10.0.2.10.6633: Flags [.], ack 111, win 229, options [nop,nop,TS val 288
4953984 ecr 1701659118], length 0
20:40:24.203889 IP 10.0.2.11.60404 > 10.0.2.10.6633: Flags [P.], seq 112:224, ack 111, win 229, options [nop,
nop,TS val 2884953986 ecr 1701659118], length 112: OpenFlow
        version unknown (0x04), type 0x0a, length 112, xid 0x00000000
```

Fig. 4.48 Comunicació entre N1 (OVS) i N5 (Ryu)

```
root@n1:/tmp/pycore.37529/n1.conf# ovs-ofctl dump-flows ovsbr0
 cookie=0x0, duration=176.341s, table=0, n_packets=5, n_bytes=434, priority=1,in_port=eth2,dl_src=00:00:00:aa
:00:53,dl_dst=00:00:00:aa:00:50 actions=output:eth0
 cookie=0x0, duration=176.335s, table=0, n_packets=4, n_bytes=336, priority=1,in_port=eth0,dl_src=00:00:00:aa
:00:50,dl_dst=00:00:00:aa:00:53 actions=output:eth2
 cookie=0x0, duration=230.384s, table=0, n_packets=50, n_bytes=3472, priority=0 actions=CONTROLLER:65535
root@n1:/tmp/pycore.37529/n1.conf#
```

Fig. 4.49 Taula de fluxos de N1 després de la comunicació amb N5

4.5. Connexió a xarxes externes

Per defecte, a CORE no hi ha connexió entre els nodes i qualsevol xarxa externa, incloent el propi host d'emulació. CORE disposa de dos opcions per connectar les topologies a xarxes externes:

- Control networks és una eina que permet connectar al host d'emulació. Aquesta eina dona accés des dels diferents nodes emulats a CORE al host on s'executa l'emulador. CORE crea un bridge al host, on hi connecta tots els nodes emulats a través dels veth. És una eina útil per executar aplicacions X11 en els nodes, per exemple.

- RJ45 és un node disponible als escenaris de CORE que representa una interfície del host d'emulació. Quan es crea el node RJ45 i s'assigna a una interfície del host d'emulació, aquesta passa a estar sota control de CORE, i deixa d'estar disponible. Qualsevol configuració prèvia de la interfície serà eliminada, incloent l'adreça IP, i s'establirà el mode promiscu per rebre i enviar dades. Per el nostre experiment, ens proporciona més flexibilitat que Control networks.

L'objectiu d'aquest apartat serà connectar l'escenari OVS anterior a un controlador localitzat en una xarxa fora de l'emulador CORE. Tal i com hem indicat anteriorment, el nostre host d'emulació és una màquina virtual, i està connectat a una xarxa host-only (192.168.56.X). La màquina virtual tindrà 2 interfícies a la xarxa host-only (**Fig. 4.51**): una serà assignada al node RJ45 (enp0s9), i l'altre donarà connectivitat a la màquina virtual (192.168.56.101). A CORE, substituirem el host que contenia el controlador Ryu per un router, al qual assignarem la IP 192.168.56.103 per connectar a la xarxa host-only i serveis de forwarding (**Fig. 4.50**) per encaminar paquets. Al host d'emulació instal·larem el controlador Ryu. CORE crea un bridge al host d'emulació on hi connecta la interfície que pertany al extrem del veth provinent del node N5, i la interfície enp0s9, assignada al node RJ45.

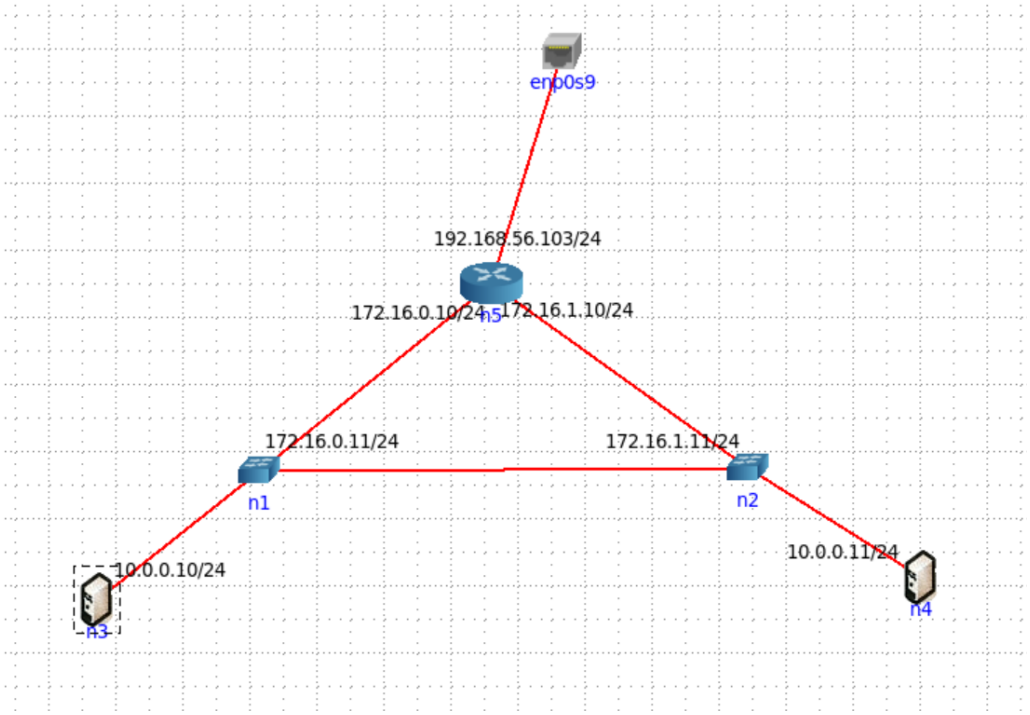

Fig. 4.50 Escenari de CORE a connectar a la xarxa externa

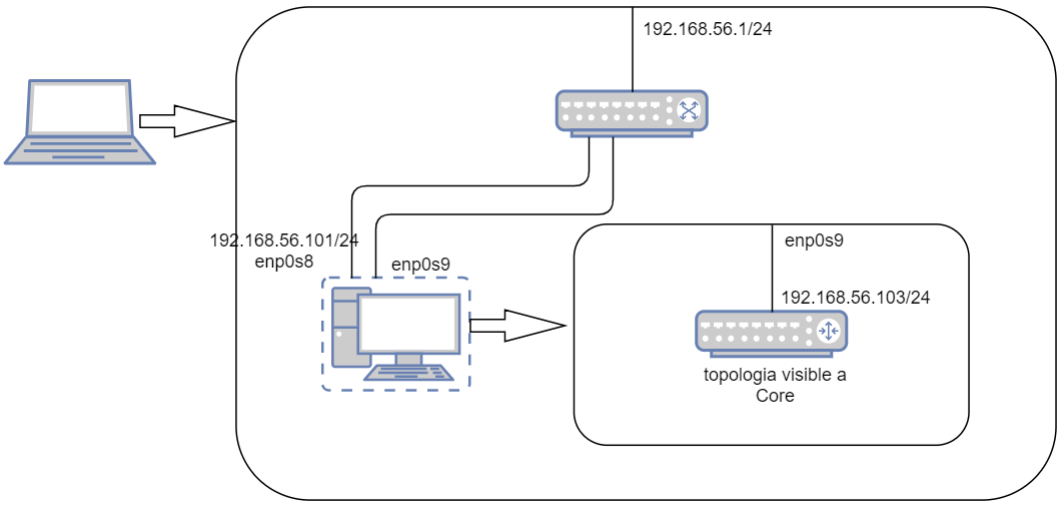

Fig. 4.51 Esquema de la infraestructura creada

Una vegada dissenyat l'entorn, l'usuari és responsable de donar connectivitat entre els diferents elements de la infraestructura elaborada. En el nostre cas, hem de donar accés des del host d'emulació als nodes OVS de la topologia i viceversa, per tal de tenir connectivitat amb el controlador Ryu (**Fig. 4.52**). S'han modificat les IP's dels enllaços entre els nodes OVS i el router N5 per un conflicte amb les IP's utilitzades al host d'emulació.

```
core@core:~$ sudo route add -net 172.16.0.0/16 gw 192.168.56.103
[sudo] password for core:
core@core:~$ route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         _gateway        0.0.0.0         UG    100    0        0 enp0s3
10.0.2.0        0.0.0.0         255.255.255.0   U     0      0        0 enp0s3
_gateway        0.0.0.0         255.255.255.255 UH    100    0        0 enp0s3
172.16.0.0      192.168.56.103  255.255.0.0     UG    0      0        0 enp0s8
192.168.56.0    0.0.0.0         255.255.255.0   U     0      0        0 enp0s8
192.168.56.0    0.0.0.0         255.255.255.0   U     0      0        0 enp0s9
core@core:~$
```

Fig. 4.52 Taula de routing del host d'emulació

Ryu incorpora una eina per visualitzar les topologies creades, amb la qual podem comprovar la connectivitat del controlador amb els dos switchs creats a CORE (**Fig. 4.53**).

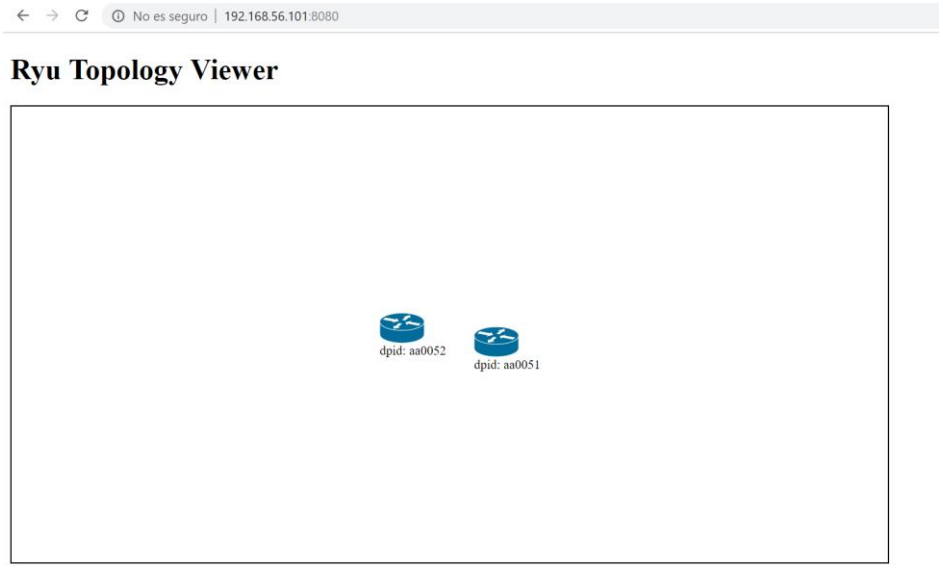

Fig. 4.53 GUI del controlador Ryu

CAPÍTOL 5. Conclusions

En aquest treball hem estudiat els blocs fonamentals de l'arquitectura de gran part del emuladors basats en software lliure, així com les seves funcionalitats més rellevants. En concret, CORE ens ha servit per analitzar els diferents casos d'ús i, documentant el procés, explorar noves facetes de l'emulador.

És obvi que el principal punt fort de CORE és l'emulació de xarxes a petita escala, basades en protocols de comunicació tradicionals, utilitzant la GUI. La GUI és amigable i pràctica en aquest àmbit, proporcionant una eina útil per desplegar i interactuar amb les topologies. Integra fàcilment nous serveis i permet personalitzar múltiples opcions de visualització, sent ideal en la formació d'estudiants i professionals que s'inicien en el món de les xarxes telemàtiques. En aquest treball hem desplegat una xarxa basada en OSPF i BGP, amb múltiples nodes distribuïts en dos sistemes autònoms. La configuració de la topologia es fa directament sobre el serveis utilitzats, integrats a la GUI, i el comportament de la xarxa, tan en la formació d'adjacències com en l'encaminament de paquets, és l'esperat.

El segon punt tractat és la configuració centralitzada i el desplegament de topologies des de CLI. Mitjançant l'arxiu IMN, hem desplegat la primera topologia amb certes modificacions, introduint certs paràmetres d'enllaç i canvis de configuració en els protocols d'encaminament. A través del CLI, hem pogut interactuar amb l'escenari i comprovar el comportament amb els canvis introduïts. Per tal de fer les mesures, hem desplegat l'aplicació IPerf als nodes, la qual ens ha permès determinar el nou ample de banda. En aquest punt ens hem familiaritzat amb dos eines clau de Linux utilitzades a CORE, Traffic Control i NetEm. Seria convenient incloure com interactuar amb aquestes eines a la documentació oficial de CORE.

CORE té una assignatura pendent com a eina per desplegar xarxes a gran escala. L'emulador inclou una sèrie de comandes per tal de gestionar els nodes d'una xarxa des de CLI, però no inclou comandes per gestionar altres aspectes fonamentals d'una xarxa, tals com la connectivitat i els enllaços entre nodes. Aquest darrer aspecte resta valor a l'emulador, ja que l'usuari segueix necessitant entendre la implementació de cadascuna de les diferents eines de l'arquitectura, tals com OVS o TC. Integrar totes les eines que formen una xarxa a CORE, en un sol paquet de comandes, restaria complexitat al desplegament de grans topologies, i faria de l'emulador una alternativa real a Mininet. A partir d'aquí, el següent pas seria incloure topologies predefinides com les que hem creat en aquest treball.

A més, el paquet de comandes per gestionar els nodes des de CLI és pobre respecte l'eina oficial lxc-tools, ja que no inclou opcions tan bàsiques com la possibilitat d'accedir al node directament, sent possible només llançar comandes des del host d'emulació. Tenint en compte que els logs dels nodes són limitats, l'experiència d'usuari per tractar des de CLI amb aplicacions desplegades als nodes no és del tot satisfactòria.

Utilitzant veth, OVS i Linux bridge com a eines de connexió entre nodes, TC i NetEM per establir paràmetres d'enllaç, i CORE per aixecar i interactuar amb els contenidors Linux, hem desplegat dos topologies des de CLI, sent capaços d'aixecar fins a 1000 nodes amb connectivitat entre ells.

És cert que CORE està fent avanços en la seva modernització, incloent serveis SDN basats en OpenFlow. L'emulador encara no inclou documentació al respecte, per la qual cosa hem volgut desplegar i documentar diversos escenaris basats en OpenFlow. Per tal de fer funcionar els escenaris, vam fer diverses probes, les quals ens van permetre entendre la particularitat de CORE en aquestes xarxes: el servei OVS és desplegat en un espai de noms aïllat, igual que la resta de nodes. Seria adient incloure, en futures actualitzacions, el servei OVS només al espai de noms root per connectar nodes, tal i com hem fet en l'apartat **4.3**, per una experiència més pràctica i escalable.

Finalment, seguint en la línia dels escenaris basats en OpenFlow, hem connectat la nostra topologia a la xarxa local del nostre entorn virtual. CORE disposa de diferents opcions per connectar a xarxes externes, característica clau dels emuladors de xarxa, per tal de ser utilitzats com a complement o extensió d'un laboratori o qualsevol xarxa física.

5.1 Futures línies de treball

L'apartat **4.3** pot ser un punt de partida per futurs treballs, marcant un repte interesant: crear un emulador de xarxa. Es tracta de combinar diferents eines per aixecar nodes, establir enllaços i proporcionar connectivitat. Partint d'eines com les que hem vist en acció en aquest treball, es pot substituir la part de creació de nodes per l'eina oficial de Linux containers (LXC), o per qualsevol altre eina per aixecar nodes, i integrar tots els components en un nou emulador de xarxes. A partir d'aquí, el repte obre múltiples camins per futurs treballs, com la creació d'un paquet basic de comandes propi per desplegar topologies o una GUI, així com incloure progressivament noves funcionalitats.

CAPÍTOL 6. Bibliografia

[1] Emulation or virtualization: What's the difference?, Dell Blog
https://blog.dell.com/en-us/emulation-or-virtualization-what-s-the-difference/

[2] Understanding Full Virtualization, Paravirtualization, and Hardware Assist. VMware
https://www.vmware.com/techpapers/2007/understanding-full-virtualization-paravirtualizat-1008.html

[3] Linux Containers (LXC) wiki, Debian SO
https://wiki.debian.org/LXC

[4] Libvirt wiki
https://wiki.libvirt.org/page/Main_Page

[5] KVM documents, Linux-kvm (by Red Hat OpenShift)
https://www.linux-kvm.org/page/Documents

[6] QEMU version 4.1.0 User Documentation
https://www.qemu.org/documentation/

[7] Docker Documentation
https://docs.docker.com/

[8] OpenvSwitch documentation (Linux Foundation)
https://docs.openvswitch.org/en/latest/

[9] Quagga documentation, Kunihiro Ishiguro
https://www.quagga.net/docs/quagga.pdf

[10] FreeBSD Manual Pages
https://www.freebsd.org/cgi/man.cgi

[11] Linux man pages online (maintained by Michael Kerrisk)
http://man7.org/linux/man-pages/index.html

[12] GNS3
https://gns3.com/

[13] Integrated Multiprotocol Network Emulator/Simulator (IMUNES)
http://imunes.net/

[14] Common Open Research Emulator (CORE). Comparison of CORE Network Emulation Platforms, Proceedings of IEEE MILCOM Conference, 2010, pp.864-869.
https://www.nrl.navy.mil/itd/ncs/products/core

[15] Mininet

http://mininet.org/

[16] Cloonix
http://clownix.net/

[17] Virtual Networks over linuX (VNX)
https://web.dit.upm.es/vnxwiki/index.php/Main_Page

[18] OpenFlow Switch Specification, Open Networking Foundation
https://www.opennetworking.org/wp-content/uploads/2014/10/OpenFlow-spec-v1.3.2.pdf

ANNEX 1

```
#!/bin/bash
vnoded -c /tmp/n1.ctl -l /tmp/n1.log -p /tmp/n1.pid
ip link add name n1.br type veth peer name n1.node
ip link set n1.node netns `cat /tmp/n1.pid`
vcmd -c /tmp/n1.ctl -- ip link set lo up
vcmd -c /tmp/n1.ctl -- ip link set n1.node name eth0 up
vcmd -c /tmp/n1.ctl -- ip addr add 172.16.0.2/16 dev eth0
vcmd -c /tmp/n1.ctl -- tc qdisc add dev eth0 root handle 1: tbf rate $2bit burst 32kbit latency 1000ms
vcmd -c /tmp/n1.ctl -- tc qdisc add dev eth0 parent 1:1 handle 10: netem delay $3ms
brctl addbr br1
ip link set br1 up
brctl addif br1 n1.br
ip link set n1.br up
y=0
z=3
for ((i=2;i<=$1 && y<256;i++))
    do
        brctl addbr br$i
        ip link set br$i up
        ip link add name br$i.int.left type veth peer name br$((i-1)).int.right
        ip link set br$i.int.left up
        ip link set br$((i-1)).int.right up
        brctl addif br$i br$i.int.left
        brctl addif br$((i-1)) br$((i-1)).int.right
        vnoded -c /tmp/n$i.ctl -l /tmp/n$i.log -p /tmp/n$i.pid
        ip link add name n$i.br type veth peer name n$i.node
        ip link set n$i.node netns `cat /tmp/n$i.pid`
        vcmd -c /tmp/n$i.ctl -- ip link set lo up
        vcmd -c /tmp/n$i.ctl -- ip link set n$i.node name eth0 up
        vcmd -c /tmp/n$i.ctl -- ip addr add 172.16.$y.$z/16 dev eth0
        z=$((z+1))
        if (($z>254))
        then
            y=$((y+1))
            z=2
        fi
       vcmd -c /tmp/n$i.ctl -- tc qdisc add dev eth0 root handle 1: tbf rate $2bit burst 32kbit latency 1000ms
       vcmd -c /tmp/n$i.ctl -- tc qdisc add dev eth0 parent 1:1 handle 10: netem delay $3ms
       brctl addif br$i n$i.br
       ip link set n$i.br up
    done
brctl show
```

Fig. 4.54 Script per la creació de la topologia lineal

ANNEX 2

OSPF

OSPF (Open Shortest Path First) és un protocol d'encaminament interior (IGP). OSPF utilitza informació d'estat d'enllaç per prendre decisions, fent càlculs mitjançant l'algoritme de ruta més curta (SPF, també anomenat algoritme Dijkstra). Els routers OSPF comparteixen anuncis d'estat d'enllaç (LSAs), que contenen informació sobre les interfícies i les mètriques d'encaminament del router.

A mesura que la informació sobre l'estat d'enllaç es comparteix entre els routers OSPF, cada router crea i manté una base de dades d'estat d'enllaç (LSDB). Els routers OSPF utilitzen la informació proporcionada dins dels LSAs com a entrada de l'algoritme SPF, calculant la ruta més adequada per a cada destí.

Paquets OSPF

- Hello. Per tal d'establir i mantenir adjacències amb veïns OSPF, tots els routers envien paquets hello per tots els enllaços cada cert interval de temps (10 segons per defecte). Aquests paquets s'envien a l'adreça multicast 224.0.0.5, que representa tots els routers OSPF, i inclou informació especifica de l'enllaç per poder establir adjacències.
- Database description. OSPF utilitza aquests paquets durant la formació d'adjacències. Serveixen per escollir el responsable de la sincronització LSDB (master), i per transmetre les capçaleres LSA (Link-State Advertisements). El router amb un identificador major és escollit master.
- Link-state request. Després d'haver rebut diversos paquets DD, un router pot determinar si una capçalera de LSA no es troba a la seva base de dades. Per tal de sol•licitar aquesta informació, el router emet un LSR.
- Link-state Update. Un router OSPF envia paquets d'actualització de l'estat d'enllaç en resposta a un paquet de sol•licitud d'estat d'enllaç (LSR), i per informar sobre un canvi en l'enllaç.
- Link-state Acknowledgment. Un router OSPF envia paquets LSA després de la recepció d'un paquet d'actualització d'estat d'enllaç (LSU).

Formació d'adjacències

L'adjacència garanteix que dos routers es coneguin i puguin estar d'acord amb certs paràmetres de l'enllaç. Aquest acord garanteix que les dades es poden transmetre de manera fiable.

Per evitar problemes de càrrega en xarxes multiaccés, OSPF selecciona un router designat (Designated Router o DR) i un router designat de backup (Backup Designated Router o BDR) que s'encarreguen d'informar a tots els routers del enllaç de les actualitzacions que es van produint, i que afecten al contingut de la base de dades. La resta de routers de l'enllaç només formen adjacència amb el DR i el BDR. En l'elecció del DR es tria el router amb major prioritat i, en cas d'empat, s'escull com a DR el que té un identificador major.

Àrees

Per tal de reduir la mida de les LSDB, OSPF permet segmentar un AS (Autnomous System) en àrees. Aquest mecanisme facilita el creixement i l'escalabilitat de les xarxes. Els LSAs queden restringits per aquesta divisió, reduint així la mida de les LSDB. OSPF manté un àrea especial anomenada backbone o àrea 0, a la qual connecten tota la resta de les àrees del AS.

Tipus de LSAs

• Router (tipus 1): descriuen les interfícies i els veïns de cada router OSPF per a la resta de l'àrea.
• Network (tipus 2): enviats pel DR a la resta de l'àrea, descriuen un segment Ethernet.
• Summary (tipus 3): enviats pel ABR (Area Border Router), que s'encarrega de connectar un àrea amb el backbone, proporcionen informació sobre un àrea per a la resta de l'AS.
• ASBR Summary (tipus 4): proporcionen informació sobre el router ASBR (Autonomous system boundary router), que delimita amb un altre AS.
• External (tipus 5): enviats pel ASBR, descriuen prefixos IP procedents d'altres protocols.

BGP

BGP (Border Gateway Protocol) és un protocol d'encaminament entre sistemes autònoms (AS). BGP admet dos tipus diferents d'intercanvis d'informació d'encaminament. Els intercanvis entre ASs són coneguts com a BGP extern o sessions EBGP. Els intercanvis dins d'un AS són coneguts com a sessions internes de BGP o IBGP.

Formació d'adjacències

A diferència d'altres protocols dinàmics, a BGP cal definir manualment els veïns. BGP utilitza TCP com a protocol de transport (port 179).

- Idle. Estat inicial.
- Connect. En aquest estat, BGP es troba a l'espera de la connexió del protocol de transport. Si la connexió es completa, el router envia un missatge Open i canvia a l'estat OpenSent.
- OpenSent. BGP espera el missatge Open del seu veí. Si es rep sense erros, envia un missatge KeepAlive.
- OpenConfirm. BGP es troba a l'espera del missatge KeepAlive o Notification. Si no es rep KeepAlive durant el temps estipulat, s'envia un Notification i es canvia a l'estat Idle. Si es rep un KeepAlive, es canvia a Established.
- Established. BGP pot intercanviar KeepAlive, Notification i Updates. Els missatges Update descriuen un camí i tots els prefixes als quals es pot arribar a través d'aquest. BGP utilitza aquesta informació per determinar el millor camí per a cada prefix.

Atributs BGP

Els atributs BGP són intercanviats en els missatges Update, i descriuen una sèrie de paràmetres per tal de determinar la millor ruta a un prefix. Alguns dels més importants són:

- Next-hop. Adreça IP del veí que adverteix el prefix. El router receptor ha de poder arribar a aquesta adreça per considerar la ruta activa. Es modifica en connexions EBGP.

- Local preference. Determina el camí preferent de sortida del AS. És un paràmetre local del AS. Valors grans tenen preferència.

- AS path. Indica el camí d'ASs per assolir un prefix. És utilitzat com a mecanisme de prevenció de bucles.

- Origin. Aquest paràmetre és introduït pel router que afegeix la ruta a BGP, i inclou informació sobre l'origen de la ruta. IGP (valor 0), EGP (predecessor de BGP, valor 1) o incomplet (valor 2), són les diferents opcions.

- MED (multiple exit discriminator). En el cas de diversos enllaços entre dos AS, s'utilitza per influenciar el tràfic d'entrada. Valors baixos tenen preferència.

Selecció de rutes

1. Valors grans de local preference.
2. AS paths més curts
3. Origin més baix
4. MED més baix
5. EBGP abans que IBGP

OpenFlow

OpenFlow és un estàndard obert que permet controlar el trànsit i executar protocols experimentals en una xarxa mitjançant un controlador remot. Els components d'OpenFlow són el controlador, el switch controlat per OpenFlow i el protocol OpenFlow. El protocol OpenFlow és un protocol de capa 2 que permet a un controlador accedir al pla de dades d'un switch OpenFlow mitjançant una connexió SSL o TCP/IP.

Amb OpenFlow es pot controlar la commutació de paquets d'una xarxa creant, suprimint i modificant els fluxos de cada dispositiu al llarg d'un camí. Les entrades de flux especifiquen les condicions de concordança amb les que es comparen els paquets, i un conjunt d'accions (OpenFlow v1.0) o instruccions (OpenFlow v1.3.1) que s'apliquen als paquets que coincideixen. Si no es troba coincidència en una taula de flux, el resultat depèn de la configuració. Per exemple, el paquet es pot reenviar al controlador sobre el canal OpenFlow. Les accions, en cas de concordança amb una entrada de la taula de flux, poden ser l'encaminament cap a un cert port del switch, modificacions de paquet o redirigir el paquet a un grup (group tables) per processament addicional.

Missatges OpenFlow

El protocol OpenFlow admet tres tipus de missatges: controller-to-switch, asynchronous i symmetric. Els missatges controller-to-switch són iniciats pel controlador, i s'utilitzen per gestionar o inspeccionar directament l'estat del switch. Els missatges asíncrons els inicien els switchs, i s'utilitzen per actualitzar el controlador amb d'esdeveniments de la xarxa, així com de qualsevol canvi d'estat del switch. Els missatges simètrics són iniciats pel switch o pel controlador, i s'envien sense sol·licitud.

Controller-to-Switch

- Features. El controlador pot sol·licitar informació sobre el switch, com identitat o capacitats.
- Configuration. El controlador sol·licita i estableix certs paràmetres de configuració del switch.
- Modify-state. Permeten al controlador modificar la taula de flux del switch.
- Read-state. Permeten al controlador recopilar informació d'estat, configuració i estadística sobre el switch.

- Packet-out. Permeten al controlador enviar paquets per un cert port del switch, així com establir decisions de commutació en resposta a un missatge Packet-in.

Asynchronous

- Packet-in. Permeten transferir el control sobre el paquet al controlador.
- Flow-removed. Informen al controlador sobre la retirada d'una entrada de la taula de flux al switch.
- Port-status. Informen sobre un canvi en un port del switch.
- Error. Informació d'errors al switch.

Symmetric

- Hello. Intercanviats pel controlador i el switch al iniciar la connexió.
- Echo. Permeten verificar la connexió, i fer mesures d'aquesta.